15

MAR 07



analisis

SUPER OFF ROAD CATWALK

ensamblador

ENSAMBLADOR DEL Z80 (II)

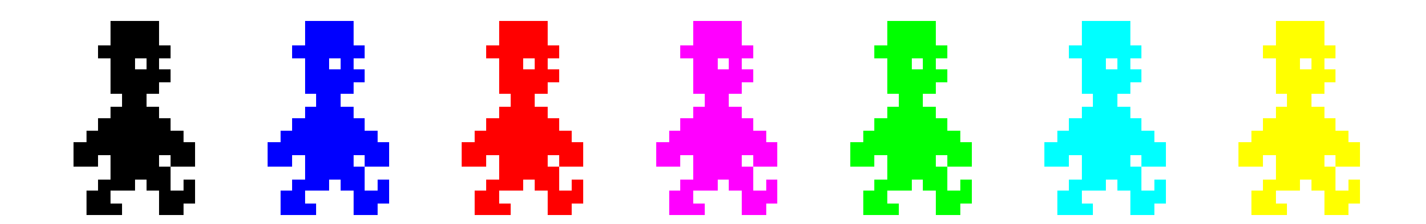
hardware

SALIDA DE VIDEO COMPUESTO EN UN SPECTRUM 48K

rem

DE LA A A LA Z DE SOKOBAN

http://magazinezx.speccy.org



De cara a la inminente MadriSX, hemos hecho un pequeño esfuerzo por acercaros un número más de la revista. No se trata de nuestra entrega más extensa, pero los contenidos que os presentamos están trabajados con el mismo cuidado que tratamos de poner desde que naciera, allá por julio de 2003.

Con esa misma ilusión es con la que compartimos con vosotros, desde un punto de vista jocoso, el proceso de creación de Sokoban. Pero nuestro entrañable robotito no estará solo.

Para empezar, hacemos un somero repaso a la actualidad de los últimos meses, en los que, día sí, día también, andamos de aniversario en aniversario.

Santiago Romero y Javier Vispe comentan sendos juegos, Super Off Road y Catwalk, respectivamente.

Hacía tiempo que no cacharreábamos en las entrañas de nuestro querido Spectrum, así que José Juan Ródenas nos envía un artículo sobre cómo incorporar una salida de vídeo compuesto al gomas.

Por último, no podía faltar una nueva entrega del curso de Ensamblador, esta vez centrada en las operaciones de desplazamiento de memoria, manipulación de bits y operaciones lógicas.

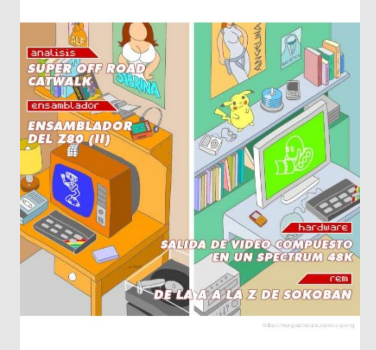
No sabemos cuándo, pero seguro que volveremos. Y nuevamente os recordamos que la dirección de correo electrónico de contacto está abierta a todo tipo de mensajes, sugerencias y colaboraciones que queráis enviarnos.

Y ahora a disfrutar todos de ese viaje en el tiempo que es MadriSX & Retro.

Redacción de MAGAZINE ZX







Año V nº15 Mar 07

Redacción

Miguel A. García Prada

Javier Vispe Mur

Federico Álvarez

Ilustración de Portada

Juanje Gómez

Colaboraciones en este número

Santiago Romero

José Juan Ródenas

Maquetación en PDF

Javier Vispe

Contacto
magazinezx@gmail.com

índice

panorama

CAMISETA "25 YEARS"

Ya lleva con nosotros 25 años esa cajita negra con teclas y un arco iris serigrafiado en la carcasa. Speccy.org lo ha querido celebrar con la edición de una camiseta conmemorativa que nos permita lucir nuestros colores. Desde aquí agradecemos a toda la gente el apoyo que han dado al proyecto. ¡Qué disfrutes la tuya mucho tiempo!



11 AÑOS DE WOS

No sólo cumple años el Spectrum. A The World of Spectrum casi ya le salen granos en la cara. Martijn y sus colaboradores continúan su labor de preservación, con una base de datos cada vez más completa y eficiente. Nada escapa a sus ojos: juegos, revistas, libros, instrucciones, mapas...

NOTICIAS DE SPA2

Sigamos con webs que son referentes en el mundo del Spectrum. Aunque ya hace algún tiempo de la última actualización de la página, no creáis que el proyecto se ha parado.





El equipo de SPA2 sigue trabajando en la sombra, recopilando material y ampliando la base de datos. Además, se está realizando un profundo lavado de cara a su diseño. Todavía queda mucho por catalogar, así que, si dispones de cintas o juegos sin preservar, colabora con SPA2. Muchos ya lo han hecho, y todos lo disfrutamos.

RETROEUSKAL'07

Esta reunión veraniega celebrada en Bilbao ya ha puesto en marcha la maquinaria para la edición de este año. Como siempre, prometen mucho trabajo para seguir aumentando la calidad y cantidad de actividades relacionadas con los sistemas retro. Este año, el Spectrum tiene algo que decir. En la actualidad, están buscando responsables para alguna de las actividades que organizan, así que es tu oportunidad de colaborar. No dudes en visitar su web y los foros.



DIVIDE

En MadriSX '06 pudimos ver en acción este interface para dispositivos IDE gracias a Julio Medina. Si te quedaste con ganas de tener uno, a lo largo de 2006 han surgido dos iniciativas particulares para fabricar pequeñas tiradas de este interface. Papaya labs está preparando las 50 unidades de su primer pedido. Rwapsoftware ha abierto las reservas para su versión, la cual permite incluso conectar otros periféricos.

En relación con divIDE, hemos de hacer notar que DEMFIR, el software que permite cargar los juegos a partir de ficheros de emulador, sigue actualizándose e incluso ya tiene soporte preliminar para archivos tzx.

PLUS3E Y RESIDOS ACTUALIZADOS

Garry Lancaster ha actualizado su proyecto de hardware y su extensión de Basic para Spectrum.

Las eproms para el plus3e llegan a la versión 1.30, dando soporte a dispositivos con tarjetas SD y solucionando algunos fallos previos.

Las mejoras de ResiDOS, que llega a la versión 1.92, se centran en una mejora de la gestión de snapshots, así como una mejora en la velocidad de lectura desde disco duro y resolución de diferentes errores.

SJASMPLUS

Bajo este acrónimo se esconde un nuevo ensamblador cruzado que permite generar código para nuestros Spectrum. Soporta todas las directivas importantes y alguna que otra inesperada como DEVICE, que permite direccionar más alla de 64K, y SAVESNA. La última versión disponible es la 1.07 RC4bf.

PUBLICACIONES

Andrew Rollings ha finalizado su repaso a la historia del software del Spectrum en forma

de libro: "The ZX Spectrum Book - 1982 to 199x ". Su trabajo se compone de 256 páginas a todo color, con referencias a las principales revistas británicas especializadas, y con prólogo de Sir Clive Sinclair.

Por otra parte, siguen apareciendo nuevas entregas de publicaciones en el panorama nacional, como ZX Spectrum Files, y en el internacional con ZX Shed.

EMULADORES DE SPECTRUM

En los últimos tiempos, las consolas portátiles se están llevando el gato al agua en este apartado:

Metalbrain continúa perfeccionando su emulador para GP2X, llegando hasta la versión 1.3.

En Nintendo DS ya son dos los emuladores que podemos usar, SpeccyDS y DSPec. Todavía falta tiempo para poder acceder a los modelos de 128K, pero sus autores ya están trabajando en ello.

En el caso de Dreamcast, Chui sigue aumentando las prestaciones de ZX4ALL, gracias a implementar el "FAZE z80 core" que ha programado Fox68K.

Por último, nos queda hacer un repaso a la actualidad en pc. Los proyectos más inquietos son SPIN y Eightyone, ambos volcados de lleno en implementar el divIDE. El segundo, además sigue mejorando el soporte de lectura/escritura de las Compact flash para Spectrum.

TAPIR

La edición de ficheros TZX con Taper no es tarea fácil en Windows XP. Gracias a un checo, Mikie, ahora disponemos de Tapir, una utilidad con casi las mismas funcionalidades que la herramienta de MS-DOS. Su desarrollo sigue en marcha, por lo que en un futuro podría incluir también un módulo para generar TZX a partir de ficheros de audio.

En paralelo al lanzamiento de Tapir, se dio por finalizada la revisión 1.20 del formato TZX

WINSGD

SGD (Spectrum Games Database) es un programa para catalogar software de Spectrum creado por Martijn Van der Heide. Hace tiempo que había dejado de ser actualizado, por lo que daba problemas bajo Windows 2000/XP. Finalmente, y tras un año de espera tras ser anunciado, Martijn Groen, nos ofrece WINSGD, una actualización para seguir usando esta base de datos bajo los últimos sistemas de Microsoft.

JUEGOS

Josep Coletas

Josep, uno de los creadores de aventuras conversacionales más prolífico de nuestro país, ha revisado algunos de sus trabajos. En primer lugar, hablaremos de la edición especial de “Los vientos del Walhalla”. Acompaña al juego un manual en PDF con instrucciones, listado de premios recibidos, comentarios... También contamos con todos los capítulos de "Los Extraordinarios Casos del Dr. Van Halen" reunidos en un sólo archivo y con una fuente más legible.

Las novedades se presentan en "Código secreto Lucybel: Trilogy". El primer episodio nos llega con mejoras, acompañado de dos nuevas entregas.

Todos estos juegos están disponibles en la web del CAAD.

CEZ games studio

na_th_an ha publicado dos juegos de golpe dentro del sello silver. Se trata de Nanako in Classic Japanese Monster Castle y Phantomasa. Dos puzzles de concepción sencilla protagonizados por dos chicas de armas tomar.

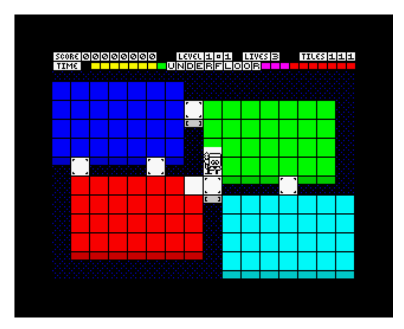
Sokoban

Compiler soft vuelve a la carga con una revisión de este clásico. En este número podéis leer un amplio artículo sobre la gestación del juego.



Demos

En el panorama intenacional, el protagonismo se lo han llevado las demos. Jonathan Smith anunció su vuelta al Spectrum con una previa de Saucer, un juego de acción que no miente de donde ha salido. En los foros de WOS, hemos visto los progresos realizados por Stranded y Bipboi.



CONCURSOS

Fiel a la cita de cada año, The comp.sys.sinclair Crap Games Competition 2006 nos ha propuesto dejar las consolas de última generación para probar los peores bodrios que han enviado sus participantes. Curiosa manera de demostrar que no se necesitan millones de euros para crear un juego malo.

Bytemaniacos ha seguido

organizando el ya veterano Concurso de juegos en Basic. En esta ocasión ha contado con tres categorías, siendo la última en llegar el reto de programar un juego en 10 líneas de código.

JAVIER VISPE

The World of Spectrum: http://www.worldofspectrum.org/

SPA2: http://spa2.speccy.org/

Retroeuskal: http://www.retroeuskal.org/

divIDE: http://baze.au.com/divide/

Plus3E: http://www.worldofspectrum.org/zxplus3e/

Rwapsoftware: http://www.rwapsoftware.co.uk/

Papaya labs: http://www.papayalabs.co.uk/

Demfir: http://demfir.sourceforge.net/

ResiDOS: http://www.worldofspectrum.org/residos/

SJASMPLUS: http://sjasmplus.sourceforge.net/

The ZX Spectrum Book - 1982 to 199x: http://zxgoldenyears.com/

ZX Spectrum files: http://microhobby.speccy.cz/zxsf/pagina_1.htm

ZX Shed: http://www.zxshed.co.uk/

GP2X Spectrum: http://www.speccy.org/metalbrain/GP2Xpectrum1.3.zip

Speccy DS: http://speccyds.wordpress.com/

DSpec: http://dspec.eighttwelve.co.uk/

ZX4ALL: http://chui.dcemu.co.uk/zx4all.html

Spin: ftp://ftp.worldofspectrum.org/pub/sinclair/emulators/pc/windows/SPIN061.zip

Eightyone: http://www.chuntey.com/eightyone/

Tapir: http://newton.sunderland.ac.uk/~mikie/

WinSGD: ftp://ftp.worldofspectrum.org/pub/sinclair/tools/pc/winsgd/

CAAD: http://www.caad.es/

CEZ game studios: http://cezgs.computeremuzone.com/

Compiler software: http://compiler.speccy.org/

CSS Crap games competition: http://www.mattrudge.net/cgc2006/

Bytemaniacos: http://www.bytemaniacos.com/





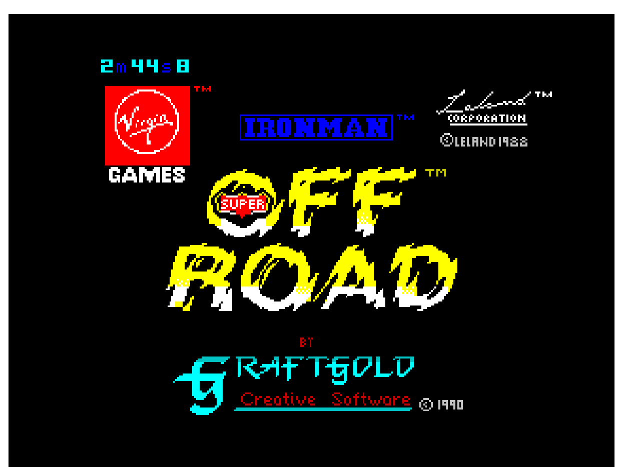
En este número, Santiago Romero analiza : Super Off Road, mientras que Javier Vispe hace lo propio con Catwalk.

SUPER OFF ROAD

Título	Ivan 'Ironman' Stewart's Super Off Road Racer
Género	Arcade
Año	1990
Máquina	48K/128K
Jugadores	1 o 2 jugadores
Compañía	Virgin games LTD
Autor	Graftgold Ltd Steve Turner John Cumming Jason Page
Otros comentarios	Your Sinclair Crash Microhobby 1 y 2



"Ivan 'Ironman' Stewart's Super Off Road Racer" es la versión Spectrum del clásico juego de los arcades donde 4 jugadores (ya sean humanos o controlados por la CPU) compiten en una carrera de circuito cerrado con coches 4x4. Tras este largo título se esconde un rápido y adictivo juego de coches patrocinado por Ivan Stewart, famoso campeón americano de carreras off-road.



Pantalla de carga de Super Off Road

Super Off Road consta de 8 circuitos diferentes que podemos recorrer en los 2 sentidos, lo que hace un total de 16 circuitos por temporada. El objetivo del juego es dar 4 vueltas completas compitiendo contra otros 3 coches, tratando de quedar primero, ya que de nuestra posición final en la carrera dependerá el dinero que ganemos y el que perdamos una "vida" (de nuestras 3 iniciales) o no.

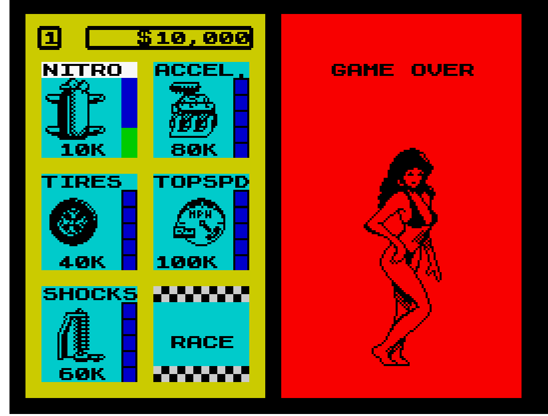
A diferencia de otros títulos como Super Sprint, aquí no competiremos en pistas de asfalto liso, sino que nos encontraremos en circuitos en mal estado, con hoyos, charcos, montones de tierra y rampas, que modificarán nuestra trayectoria obligándonos a realizar continuas correcciones a izquierda y derecha, saltos y adelantamientos con el objetivo de no quedarnos atascados en alguna esquina, agujero o accidente del terreno.


El primer circuito de Super Off Road

Cabe destacar que los choques con otros coches tienen más utilidad de la que parece: ralentizan a los coches rivales, aceleran los nuestros, nos permiten sacar a enemigos de la pista, etc. No es una mera cuestión de correr más que los rivales, también podemos tomar ventaja de los accidentes del terreno (como usar un turbo antes de tomar la rampa para saltar el río, por ejemplo) y del "combate coche a coche".

Los controles son los típicos de esta clase de títulos: acelerar, frenar, girar a izquierda y girar a derecha (relativos a la dirección en que miramos). Además, disponemos de un botón de disparo que hará que nuestro coche gaste un NITRO o TURBO (de la reserva que dispongamos), impulsándolo a más velocidad de la normal durante un tiempo limitado. Estos controles pueden ser Joystick Kempston o Sinclair, los cursores, o las teclas "A Z N M ESPACIO" o "K M Z X ESPACIO".

Contamos con una cantidad de dinero inicial que nos permitirá la compra de más nitros así como mejoras en nuestro vehículo: mejoras en la aceleración (tiempo de respuesta), las ruedas (giro del vehículo), el motor (alterando la velocidad máxima de nuestro coche) y la suspensión (para el agarre y los saltos) podrán ser adquiridas a cambio de dinero.



Opciones de mejora de los vehículos

¿Y cómo obtendremos ese dinero? De múltiples formas: aunque empezamos con una pequeña cantidad inicial de dinero, podremos obtener ingresos adicionales según nuestra posición al finalizar las 4 vueltas de cada carrera: 100.000, 90.000 y 80.000 US$ para el primero, segundo y tercer clasificado respectivamente, quedando el último sin recompensa. Tendremos que quedar primeros en cada circuito para no perder ninguna de nuestras vidas.

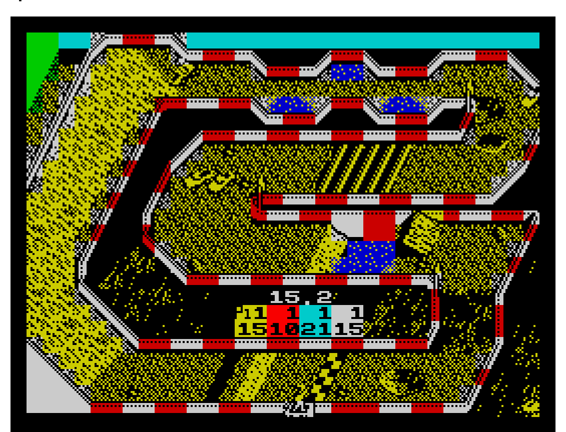
Y es muy importante el no perder vidas porque, si lo consideramos conveniente, podemos cambiarlas por 200.000 US$ (cada una) en la pantalla de mejoras. Para hacerlo, basta con comprar cualquier cosa cuando tengamos nuestro contador a 0 US$. El juego nos canjeará una vida por 200.000 US$ y comprará lo que le hemos pedido.

La otra forma de conseguir dinero (y nitros) es en la pista: durante el desarrollo de la carrera aparecerán nitros y bolsas de dinero que nos irán dando turbos y dinero adicional. Además, cada vez que se recoja uno de estos elementos, el siguiente que aparezca será mayor (los 10.000 US$ de la primera bolsa de dinero se convierte en 20.000 US$ en la segunda, 30.000 US$ en la tercera, etc).

El objetivo del juego es alcanzar el final de la temporada (16 carreras en los 2 sentidos de 8 circuitos) con la mayor cantidad de dinero ganado.

A continación justificaremos las notas que hemos otorgado al juego en Originalidad, Gráficos, Sonido, Jugabilidad, Adicción y Dificultad.

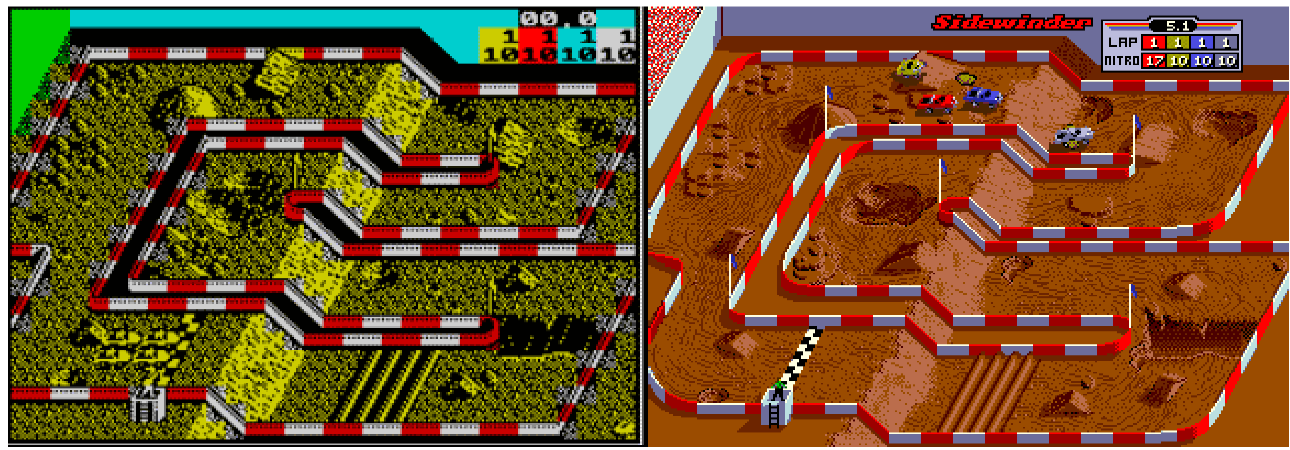
El mayor elemento de originalidad del juego es el hecho de ser uno de los primeros títulos de la historia donde se puede modificar el rendimiento del coche con añadidos y mejoras en motor, ruedas, etc. Aunque existen bastantes títulos similares de calidad, como el clásico Super Sprint, las mejoras, los terrenos con saltos, charcos, obstáculos y los nitros dotan de una personalidad especial a este juego. De hecho, es muy probable que sea uno de los títulos de este género más recordados, al haber sido portado a gran cantidad de plataformas.



Charcos, saltos y montones de tierra

Gráficamente el juego está a un buen nivel, excepto por el colorido de los vehículos. Las limitaciones técnicas de representación de color del Spectrum impiden que, como en otras plataformas, cada coche sea de un color diferente. Como en gran cantidad de títulos de Spectrum, los sprites de los coches son "transparentes" y toman el color del fondo, que suele ser "amarillo tierra" o el azul del cruce de algún charco ocasional. Esto es en ocasiones un gran problema, porque cuesta distinguir nuestro coche del de nuestros rivales, haciendo que se resienta la jugabilidad.

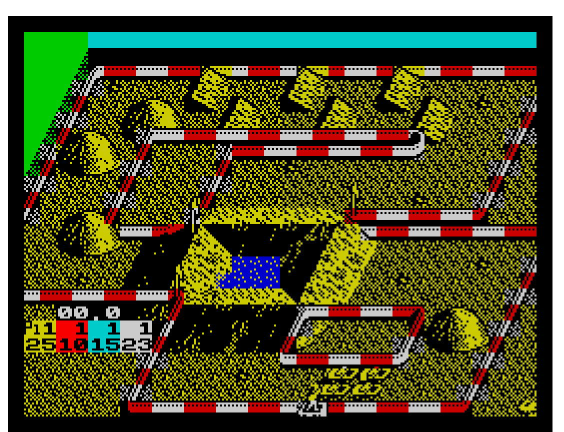
Más de una vez os encontraréis intentando averiguar cuál de los 4 coches sois vosotros, especialmente tras encontronazos o choques. Teóricamente, una pequeña banderita sobre el coche nos distingue de los rivales, pero no es fácil verla, y mucho menos en movimiento. Lo normal



Comparación Spectrum vs Arcade: colorido de los 4x4

es seguir a nuestro coche con la vista, lo cual es factible hasta que empiezan los atascos, saltos, o choques. En ocasiones incluso estaréis creyendo controlar un coche que está conduciendo a la perfección cuando, en realidad, estaréis atascados en una esquina dando vueltas. Esa sensación de descontrol mengua con el tiempo, pero puede ser algo frustrante al principio.
El control en sí es bueno y suave, los coches responden a la perfección y los saltos, turbos y coches transcurren de una manera fluida (también es cierto que el no tener scroll de pantalla ayuda a ello). En ocasiones resulta demasiado sencillo quedarse atascado en una esquina o ser desplazado por otro coche, pero eso es parte de la dificultad del juego, y lo hace bastante adictivo, especialmente a 2 jugadores (aunque no se haya podido llegar a los 3 y 4 jugadores de las versiones arcade/consolas y NES).

El nivel sonoro es aceptable; una buena música en el menú da paso a simples efectos sonoros durante el juego, que cumplen su función: turbos, recogida de objetos y choques son los que oiremos más habitualmente.


A punto de realizar la salida en el 3er circuito

La mejor baza del juego es, sin duda, su jugabilidad y adicción: un juego con un planteamiento sencillo, competición con otros vehículos, mejoras en los coches, variedad de circuitos, lucha por los objetos en el terreno de juego y esa sensación de intentar superar ese circuito en el que siempre perdemos.

No obstante, es un juego a corto o medio plazo, ya que no invita a jugarlo en monojugador durante una larga temporada. Aun así, a largo plazo los 8 circuitos en sus dos sentidos son interesantes y lo suficientemente diferentes para disfrutar este título en el modo de 2 jugadores.

LINKS

Mapa de circuitos:
ftp://ftp.worldofspectrum.org/pub/sinclair/games-maps/i/IvanIronmanStewartsSuperOffRoadRacer.png

Referencia en la WikiPedia:
http://en.wikipedia.org/wiki/Super_Off_Road

Biografia de Ivan Stewart:
http://www.protruck.com/ivan.html

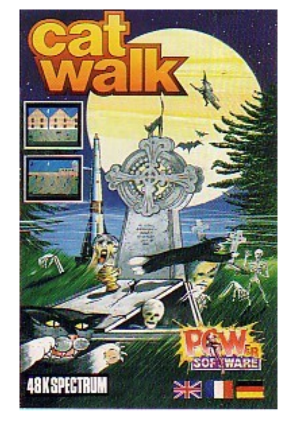
Valoraciones

originalidad	[7]	jugabilidad	[8]
gráficos	[7]	adicción	[9]
sonido	[7]	dificultad	[8]

SROMERO

CATWALK

Título	Catwalk
Género	Plataformas
Año	1984
Máquina	48K
Jugadores	1 jugador
Compañía	Power software
Autor	Paul Barsby
Otros comentarios	Crash



Conocí este juego por medio de una cinta Load 'n' Run, de esas que incluían copias piratas de juegos modificados. Sin embargo, a nosotros eso nos importaba bien poco. En un mismo soporte tenías una docena o más de juegos que te prometían horas de diversión. No siempre era así, puesto que lo de menos era cuidar la calidad de la recopilación. Sin embargo, en el caso que nos ocupa, sí puedo decir que me he pasado horas muertas delante del televisor.



Pantalla de carga

"El gato" realmente es un juego inglés que lleva por nombre "Catwalk". Nuestro objetivo es acompañar a Snooky en sus correrías nocturnas, alimentándose de ratones, pájaros y por supuesto, comida para gatos. La presentación nos muestra como salimos de nuestro hogar dispuestos a cazar en los nueve escenarios donde discurre nuestra aventura. Snooky es un gato algo especial, puesto que cuenta con 9 vidas para conseguir su objetivo. La estructura del juego es similar a Manic miner, con fases de plataformas donde debemos recoger la comida, y que superaremos tras obtener todos los objetivos. Sin embargo, en alguna de ellas podremos movernos en unas 3 dimensiones algo "sui generis".

Los escenarios empiezan en un típico barrio inglés, pero conforme se avanza, cada vez son más surrealistas, e incluso podemos llegar a entrar dentro de una base de lanzamiento de cohetes o un extraño salón arcade. Bendita imaginación inocente de los ochenta, que tanto se echa de menos hoy en día en medio del hiperrealismo extremo...

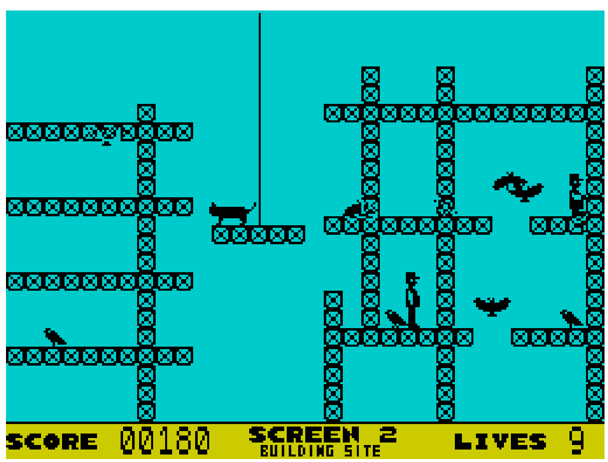


Introducción

Gráficamente el juego cumple bien. Es 1984 y todavía no han llegado las florituras. Se utiliza bastante color en algunos escenarios, y curiosamente, en otros se opta por la bicromía absoluta. Los personajes siempre visten de riguroso luto, en un intento por evitar la moda del "colour clash". Sin embargo, no se libran de ciertos parpadeos producto de una programación poco cuidada. Las animaciones de los sprites son simpáticas, y en el caso del gato, a pesar de no contar en aquellos tiempos con técnicas de "motion capture" se acerca bastante a la realidad. ¡Qué elegante en sus andares!



Snooky puede saltar y moverse en dos o cuatro sentidos dependiendo de la fase. Con el salto en ocasiones hay que tener cuidado para no acabar cayendo por los numerosos huecos que se encuentran en pantalla. Significará una muerte segura, por mucho que los gatos siempre caigan de pie. Aquí la muerte del animal no nos producirá pena, sino cierto odio al programador. Cuando pierdes una vida, Snooky sube al cielo, lo que implica tener que esperar unos cuantos segundos hasta que el puñetero gato se va hacia arriba del todo. Esta molestia es un pequeño contratiempo en nuestras ansias por seguir pasando pantallas. Podría haberlo programado un ateo...



Segunda fase

La música en el juego se compone de dos melodías del altavoz del Spectrum, en el menú de inicio, y cuando pasamos de pantalla con éxito. En cuanto a los efectos, son simples pitidos para indicar la captura de alimento, saltos o una escala descendente cuando Snooky nos somete al martirio de su ascensión a los cielos. En resumen, sonidos electrónicos de bajo coste.

La dificultad en las primeras pantallas no es muy grande, empezando a ponerse fea la cosa a partir de la factoría. Todo dependerá de nuestra pericia y paciencia para ir planeando nuestra acción. El punto fuerte del juego es la adicción que supone el reto de ir pasando pantallas. Son pocas, pero quizás por eso, nos planteamos que es asequible el acabar el juego en una tarde sin sufrir. La recompensa de pasar un nivel que se nos ha atravesado, no tiene precio.

El juego nos permite elegir tres idiomas al principio: inglés, alemán y francés. Parece que la moda de obviar el castellano en los vídeojuegos no es cosa de las videoconsolas. En cuanto a los controles, podemos usar joystick kempston o redefinir teclas.

Catwalk es un juego más que se perdió en la noche de los tiempos, entre decenas de aspirantes a robar parte de la fama conseguida por el minero Willy. Nadie se acuerda de Snooky, pero bien merece la pena darle una oportunidad para pasar unos minutos entretenidos. Un concepto de juego simple y efectivo que además nos permite ver, que, además de los perros, los gatos también van al cielo. Aunque tengamos que matarlos nueve veces.



Tercera fase

Descárgalo de WOS:

http://www.worldofspectrum.org/infoseekid.cgi?id=0000848

Valoraciones

originalidad	[6]	jugabilidad	[7]
gráficos	[5]	adicción	[7]
sonido	[4]	dificultad	[7]

JAVIER VISPE



hardware

SALIDA DE VÍDEO COMPUESTO EN UN SPECTRUM 48K

Como todos sabéis, la calidad de la señal de vídeo RF que ofrecían los primeros Spectrum (léase 16k y 48k) no estaba muy allá. Buceando por Internet encuentras páginas de gente maravillosa que intenta incorporar a aquellas máquinas toda clase de "aparatos" o útiles que los mejore de alguna forma.

Hoy os presentamos una de estas mejoras reseñadas en la página de los amigos de OCTOCOM y que pertenece a un monstruo de la electrónica, Sami Vehmaa. La aplicación se refiere a modificar la señal RF de un Spectrum e incorporarle señal RCA con lo que mejoraremos ostensiblemente la señal y podremos utilizar nuestro Spectrum en cualquiera de las televisiones actuales.

Aclaremos, no obstante, que la práctica la vamos a realizar sobre un equipo que posee un teclado dk'tronics, objeto de este artículo. La diferencia entre este montaje y lo que realizó Sami Vehmaa estriba en que lo que hemos hecho ha sido proporcionar la nueva señal sin desmontar la antigua. Vamos a ello.

Material necesario:

- Soldador y estaño.
- Pelacables (opcional).
- Transistor.
- Condensador.
- Resistencia (75ohm a 82ohm).
- Conector hembra RCA.
- Pletina de metal, arandelas y tornillo.



Imagen 1. Algunos componentes

La acotación que hacemos sobre la resistencia es debida a que en la página de Sami indica la necesidad de una resistencia de 75 Ohm, aconsejada igualmente por José Leandro Novellón, al que se le comentó la realización de la práctica. En cambio, los amigos de Octocom se deciden por una de 82 Ohm. Nosotros para no ser menos, hemos tirado por el camino de en medio y hemos utilizado lo que teníamos más a mano, soldando dos resistencias que dan un total de 78 Ohm.

LA PRÁCTICA

Tal y como indican los amigos de Octocom en su página deberemos reconocer primeramente la ubicación del modulador y los datos que nos ofrece; para ello exponemos aquí dos instantáneas sacadas de su web. Una es el esquema del montaje y la otra los puntos donde van a realizar la modificación, una vez retirado el modulador.

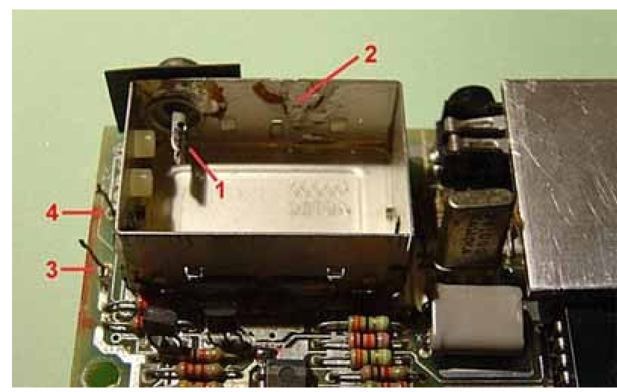


Imagen 2. Vaciado del modulador

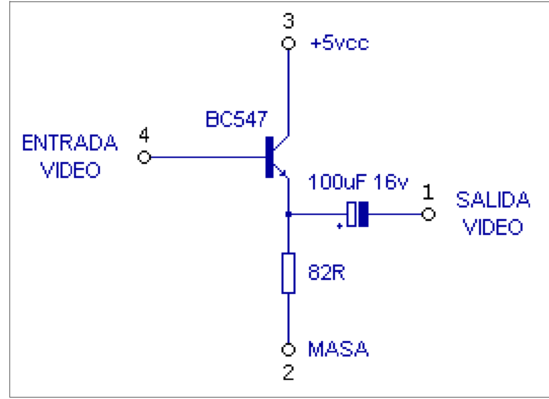
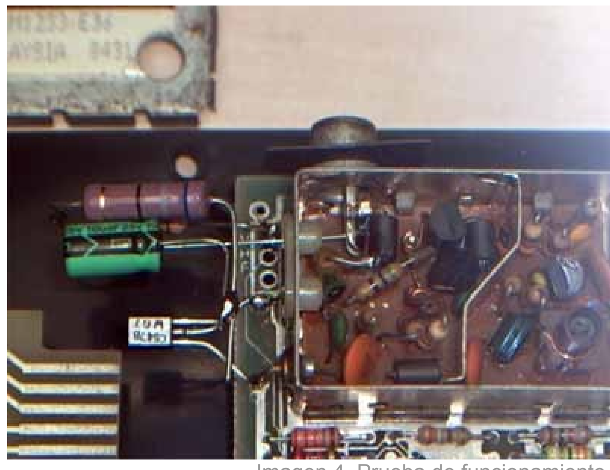


Imagen 3. Esquema de la práctica

Tomaremos buena nota de los cuatro puntos a tener en cuenta pero no quitaremos nada, aprovechando la ventaja de estar trabajando en un espacio más amplio que el que proporciona el teclado del 48k.

Si seguimos el esquema veremos que la resistencia debe soldarse a masa, la cual encontraremos fácilmente en la misma caja del modulador (punto 2, esquema en interior caja). O si por el contrario se nos resiste o no necesitamos abrir el modulador, como es nuestro caso, buscaremos en una de las pistas próximas a éste, comprobándola previamente. El otro extremo lo soldaremos al condensador.


Imagen 4. Prueba de funcionamiento

El transistor BC547 presenta tres extremos: base,emisor y colector.

Por ello, lo primero que haremos será diferenciar estas tres vías del transistor.

El transistor se alimenta por el colector, de ahí la conexión a +5V que se indica en el esquema, y necesita de la resistencia para su correcto funcionamiento. La señal entra al transistor por la base y sale amplificada por el emisor hasta el condensador; el cual sçolo permite pasar la corriente alterna, o sea, la señal.

Una vez soldada la resistencia en el emisor del transistor y al condensador, la otra pata de este llevará finalmente la señal de vídeo compuesto tan preciada. Antes de soldarla al conector RCA, hemos optado por probarla y que mejor forma que desoldando la señal RF y soldando la nueva, (imagen 4).

A continuación nos aprovecharemos de las características del teclado dk'tronics para colocar un conector hembra RCA en la salida inmediata superior a la que sale del modulador. (imagen 6)

La señal la soldaremos a un conector hembra RCA. La masa de este conector, (tal y como se ve en la imagen 5) hemos optado por soldarla en una de las pistas del exterior de la placa.


Imagen 5. Colocación del nuevo conector RCA

Finalmente la colocación del conector hembra RCA lo hemos resuelto sujetando una pletina a la tapa inferior del teclado dk'tronics haciéndole un agujero y sujetándola con un tornillo.



Imagen 6. Visión exterior

Observamos en la imagen anterior ambas salidas de vídeo RF y RCA, sin sustituir nada. Eso sí, si pretendemos incorporar el Interface 1 al teclado, deberíamos perder una de ellas, obviamente la RF, pero simplemente quitaríamos el nuevo conector y dejaríamos la práctica tal cual se ve en la imagen 4.

Ahora sólo queda enchufar el Spectrum a una de nuestras modernas televisiones y disfrutar del Uchi-mata.

CRÉDITOS

Desarrollo de la práctica: José Antonio Pérez Grau, aka 'buyer'

Artículo: José J. Ródenas - aka 'sejuan'

LINKS

http://www.octocom.es

http://user.tninet.se/~vjz762w/

J.J.RÓDENAS



DE LA A A LA Z DE SOKOBAN, O CÓMO LAMERSE EL CIPOTILLO UNO MISMO

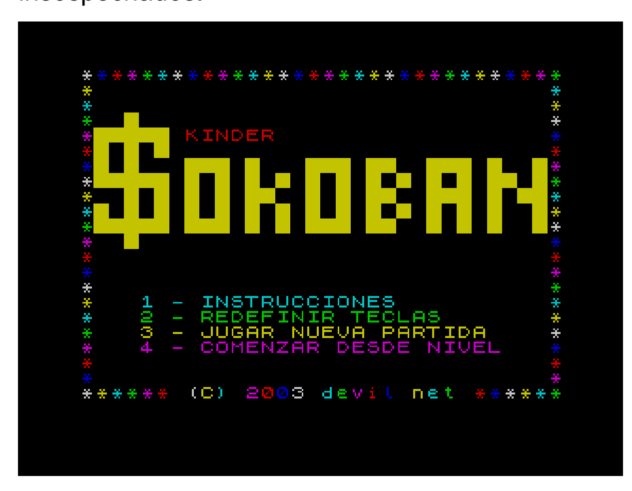
Miguel A. García prada nos cuenta...

PRIMEROS PASOS

Cuando se convocó el concurso de juegos en BASIC de Bytemaniacos, del lejano año 2003, la primera idea que me vino a la cabeza fue realizar una versión del Sokoban.

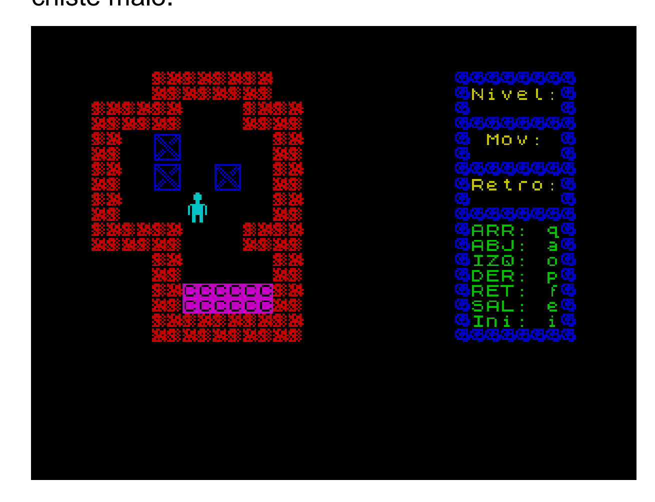
¿Por qué? Sokoban, ideado por un japonés llamado Hiroyuki, a principios de los ochenta, es uno de los juegos más versionados y adictivos que conozco. Muchas veces hace que dejes un nivel y te vayas a la cama pensando en como solucionarlo, jugando en tu cabeza con una imagen mental de la pantalla en la que te encuentras atrancado.

La idea del juego es simple: se trata de colocar unas "cajas" en determinados lugares de una habitación. Las cajas únicamente se pueden empujar, nunca tirar de ellas y no es posible empujar más de una a la vez. Con estas sencillas líneas el juego alcanza niveles de dificultad insospechados.



Después de estudiar el juego y ver que era posible portarlo al Spectrum en BASIC, por supuesto con todas las limitaciones que acarrea el lenguaje en cuestión, me puse manos a la obra con la programación.

En poco más de una semana ya se movía el personaje, un "bicharraco" con forma de muñeco del día de los inocentes que podéis ver en una captura debajo de estas líneas. Y es que los gráficos no son lo mío. El monigote empujaba las "cajas", detectaba muros, era capaz de terminar una pantalla… El programa mapeaba las pantallas y tenía un menú de opciones para comenzar a jugar. No recuerdo exactamente el motivo del "kinder" en el nombre del juego, fue una broma de Santi Romero y, conociéndole, seguro que era un chiste malo.



Pero si de una virtud carezco es de la perseverancia. Y cuando el juego estaba en un 85% de desarrollo aproximadamente, lo abandoné, y no se presentó a concurso. Y Así quedó la cosa hasta los comienzos del año 2006.

SEGUNDO INTENTO

A primeros del 2006 me propuse hacer, y terminar, un juego para Spectrum, pero con la particularidad de que quería programarlo íntegramente en ensamblador de Z80, una pequeña espinita que tenía clavada desde los años ochenta. Y me decidí a retomar el Sokoban.

De inicio comencé en solitario con el proyecto: gráficos, programa, etc. Pero un par de decisiones, bastante acertadas, le dieron un giro de 180º a Sokoban.

Primero se lo comenté a Javier, que se puso a hacer gráficos a la velocidad de la luz. Según le enseñaba betas, me contestaba al correo con gráficos. Le dio la vuelta al juego en el aspecto visual. Fede aceptó hacer la música, algo en lo que yo estaba bastante perdido. Y con esto puede dedicarme únicamente a programar el código del juego.

Después me dediqué a bombardear, literalmente, a un grupo de amigos del mundillo con betas y

más betas del juego. Lejos de mandarme a buscar gamusinos al bosque, empezaron a dar ideas, consejos y sugerencias.

Mi idea inicial del juego era darle un estilo más "arcade" que el original, con la inclusión de parámetros como tiempo y "undos" limitados para cada pantalla, y según la dificultad de la misma. Una vez programado se llegó a la conclusión de que, al fin y al cabo, Sokoban es un juego para pensar, y que la presión de tener que terminar un nivel en un periodo de tiempo determinado no era positivo. Se desechó el contador de tiempo, y también el de los "undos" por nivel, en su lugar se decidió incluir la posibilidad de jugar con o sin la opción de retroceder en los movimientos.

NIVELES

Con la programación del juego bastante avanzada se planteó el reto más importante: los niveles. En un principio se iban a diseñar todos desde cero, pero según avanzábamos en ellos nos dimos cuenta de que era la parte más difícil de todo el juego. Al margen de crear la distribución de cada pantalla y hacerlos más o menos atractivos, lo más complicado es dotarlos de una dificultad ajustada. Nos pusimos a indagar por Internet y encontramos, entre otras muchas, a dos personas que se dedican a hacer niveles de Sokoban y que los ceden para su uso libremente: Evgeny Gregoriev y David W. Skinner. Después de una selección entre los cientos de niveles que tienen cada uno de ellos, se eligieron los más apropiados para nuestra versión en Spectrum, ya que, esencialmente por tamaño, no todos los niveles servían. Una vez eliminados los marcadores de tiempo y "undos" de la pantalla de juego, se decidió usar toda la superficie para zona de juego. El tamaño de los "tiles", piedras a empujar y protagonista se fijó en cuadrados de 16 por 16 píxeles, con lo que las pantalla podían tener un máximo de 16 por 12 "tiles" de tamaño, lo que reducía mucho el rango de niveles que nos servían.

La cifra de niveles se cerró en 99, de los cuales la primera docena se hizo desde cero por los diferentes colaboradores del juego, para la versión de Spectrum, el resto se tomaron de los realizados por Evgeny y David. Se decidió que los doce niveles realizados para Spectrum se pusieran al principio del juego, ya que no eran muy complicados, y así sirviesen de tutorial o aprendizaje y el resto se colocaron a continuación. La dificultad de los mismos no va en aumento según se avanza en el juego, sino que se alternan fáciles y más complicados. De esta manera te puedes tirar horas o días para superar un nivel, pero los dos o tres siguientes serán más sencillos.

PROGRAMACIÓN

El principal reto, respecto a la programación, no ha sido el ensamblador en sí, al que yo tenía miedo y que al final ha resultado ser más dócil de lo que me imaginaba y que permite hacer cosas que en BASIC ni te planteas, si conoces ambos lenguajes, por supuesto. El principal problema ha sido conseguir meter el juego en la reducida memoria del Spectrum, apenas 40Kb en el modelo "base" (dejando de lado el 16K).

Desde un principio se tenía claro que el juego tenía que correr en cualquier modelo de Spectrum: desde el 48K hasta el +3. Y por supuesto no correr sólo en emulador, ya que el fin de programar un juego de Spectrum es que corra en un Spectrum. Aun a sabiendas de que la mayoría de la gente que lo disfrute lo hará en un emulador, no saben lo que se pierden, ya que el juego funciona bastante mejor aporreando teclas de hace 20 años que el teclado de un PC. La única diferencia resultante de correr el juego en modelos que no incluyen el chip AY para generar el sonido es esto mismo, la música, que en los modelos de 48K no sonará. Pero creemos que no es obstáculo para quien quiera disfrutar del juego en una máquina con teclado de goma, que tiene su encanto.

El mapa de memoria, al final, ha quedado bastante ajustado e incluso han sobrado casi 3KB libres. Los niveles ocupan alrededor del 20% de la memoria, la música y el player sobre los 3KB, de los gráficos mejor no hablo ya que tengo pesadillas todavía al respecto. El resto está dedicado a programa y textos.

Yo soy un poco raro a la hora de ponerme a programar, suelo empezar la casa por el tejado. Lo primero que hice fue el menú, luego el mapeador de pantallas, y poco a poco se fue incorporando el movimiento, la detección de los "muros", movimiento de bloques, etc.

Uno de los apartados más espinosos para mi, en un principio, era el de implementar la parte de los "undos". Pero al final resultó ser de lo más sencillo. Una simple pila con la dirección del último movimiento realizado, hasta un total de cien, en la que vamos almacenando los mismos. Ahora, tal como se terminó el juego, sería posible aumentar esa pila hasta los dos mil movimientos almacenados sin ningún problema, pero cuando programé la función íbamos bastante ajustados en las previsiones de memoria, y se pensó que la posibilidad de retroceder cien veces era más que suficiente (otros gurús dijeron que con 640K de memoria tendríamos de sobra y vamos por varios gigas...)

El movimiento fue muy sencillo de programar ya que, tal como comenta Javier sobre los gráficos,

se decidió hacerlo de 8 en 8 píxeles para evitar el "colour clash" propio del Spectrum, y así poder dotar de todo el colorido posible a nuestro robot y a los objetos a empujar. Dado que cada nivel está almacenado en una matriz de 16 por 12, la detección de los diferentes obstáculos que podemos encontrar en las pantallas es sumamente sencilla. Simplemente navegando con un puntero por esa matriz y viendo si el código indicado es un muro, un objeto a empujar o está vacío. Esta misma matriz se utiliza para mapear el nivel en la pantalla y en ella se indica, con diferentes códigos, el tipo de bloque que se imprime, las piedras y su situación, las casas, etc.

Respecto a la música decidimos utilizar un player ya existente, el Vortex. Mis conocimientos de cómo funciona el chip AY del Spectrum son nulos y gracias a este programa me solucionó la papeleta. Lo único que programé fueron las llamadas al player, mediante interrupciones, y la opción de activar y desactivar la música en tiempo de juego. La inspiración la puso Fede, que para algo tenían que servir las horas perdidas aprendiendo solfeo y aporreando el organito Casio.

El peor quebradero de cabeza lo tuve con el sistema de claves para empezar desde el último nivel terminado, algo que se consideró imprescindible. Debe de ser muy malo para la salud terminar los 99 niveles del tirón, sin descansar. En los primeros pasos del juego los niveles estaban divididos en dos "sets" diferentes: uno con los niveles que estábamos haciendo en exclusiva para el Spectrum, y otro "set" con los niveles de David y Evgeny. Con lo cual había que almacenar el set en el que estábamos, el nivel y alguna otra información. Hay momentos en los que te atascas en algún sitio, y este fue uno de ellos, la mala utilización de una instrucción de rotado de bits hizo que, al generar una clave para un nivel superior al diez, el programa cargara cualquier cosa menos un nivel jugable, con el consiguiente cuelgue del sistema. Cuando analizas estas situaciones y las solucionas, piensas que es increíble que te puedas quedar dos días atascado en el mismo punto, pero la satisfacción al ver funcionar correctamente algo que te ha costado tanto programar es superior a la fatiga.

Realmente he tardado más en reprogramar lo programado, gracias a las sugerencias que se daban sobre la marcha, que a realizar el motor del juego en sí. Una de las cosas que mejor resultado ha dado es la ventana de opciones en tiempo de juego. En un principio, al pulsar una tecla de pausa que había, la pantalla se ponía en negro para evitar, cuando el juego funcionaba con tiempo limitado, que se hicieran trampas estudiando los movimientos. El menú en tiempo de juego le da un aspecto muy llamativo y un acabado que hubieran querido muchos juegos que se lanzaron comercialmente en los ochenta.

Los momentos más divertidos de la programación han sido realizar el final (sí, el juego tiene un divertido final al completar los 99 niveles) e incluir un "cheat" que activa un nuevo set gráfico y alguna cosilla más. Por cierto, muy sencillo de descubrir a nada que se pegue un vistazo al código fuente del juego.

El juego ha sido programado utilizando ordenadores actuales, bajo emulador, y utilizando todos los beneficios que obtenemos de la velocidad de proceso de las máquinas de que disponemos ahora mismo. Un compilado en Spectrum de un programa en ASM puede durar bastante tiempo, un cuelgue por una mala instrucción milésimas y supone una carga de minutos desde cinta de cassette, mientras que lo mismo en un PC es cuestión de segundos. Se adelanta muchísimo y no quiero ni pensar en como se las verían los hermanos Ruiz programando hace 20 años…

Las herramientas utilizadas han sido, básicamente, FUSE como emulador, PASMO como compilador (excelente compilador) cruzado de Z80 y VIM como editor para el código, todo ello bajo LINUX que era el sistema que utilizaba. En las últimas fases de la programación el editor utilizado fue ForgEdit, para MAC, ya que cambié mi ordenador por uno de Apple, aunque seguí utilizando el mismo compilador y emulador, disponibles para el ordenador de la manzana.

Y poco más que comentar, salvo que el código fuente está disponible libremente para quien le interese echar un vistazo. ASM no es el hombre del saco, se escriben más líneas, tardas más en ver resultados que en BASIC, pero merece la pena. Los resultados finales inclinan la balanza, sin ninguna duda, del lenguaje de los Z80.

Solo me queda agradecer a todos aquellos que han hecho posible mi sueño, el ver terminado un juego para Spectrum. Son muchos, y todos ellos están en los créditos del juego, por lo que no voy a repetir aquí los nombres, cual Almodóvar recitando el santoral...

A continuación, Fede J. Álvarez nos comenta...

CÓMO HACER QUE LA NOTA DE UN JUEGO NO LLEGUE AL 10

Cuando me enteré de que Miguel iba a retomar su vieja idea del Sokoban, la verdad es que me alegré mucho, ya que yo fui uno de los que siempre andaba detrás de él dando la brasa para que lo acabara. Pero las noticias realmente buenas llegaron cuando supe, por un lado, que el juego iba a ser reprogramado totalmente en ensamblador, desde cero, y que Javier Vispe andaba metido en el ajo.

Por supuesto, fue un orgullo que Miguel me

ofreciera la posibilidad de hacer la música. Ya tuve la ocasión de adaptar la sintonía de Babaliba al remake, pero esto suponía un nuevo reto, ya que se trataba, por un lado, de componer una música original y, por otro, de hacerlo para un chip que da apenas 3 canales de sonido generado digitalmente, sin posibilidad de usar *samples* de instrumentos reales.

Las primeras pruebas las hicimos con el programa Wham!, en sus dos versiones, para 48K y 128K. Tuve que familiarizarme con su funcionamiento mientras Miguel se pegaba con el *player*, tratando de integrarlo en el código de Sokoban. Pero la cosa no funcionó demasiado bien, la música no sonaba como debía y el juego se ralentizaba demasiado.

Unos meses después, el proyecto iba tomando la forma que todos hemos podido comprobar al final del todo, y Javier Vispe, impagable conocedor de los entresijos de la actualidad *spectrumera* nos facilitó el enlace de un *tracker* para Windows, llamado Vortex Tracker y programado por Sergey Bulba. Dicho *tracker*, además, incorpora un *player* para Spectrum, así que Miguel se puso nuevamente manos a la obra con la tarea de integrarlo con el resto del código. Esta vez funcionó, así que ya sólo quedaba componer la música.

Lamentablemente, estos últimos meses no he dispuesto de demasiado tiempo libre precisamente. Lo cual se ha traducido en que sólo he podido familiarizarme mínimamente con la herramienta y la composición resultante ha quedado un poco pobre. Así que me he propuesto, de cara a futuras composiciones, al menos estudiar y escuchar ejemplos de música en los que sus autores han conseguido sacarle muchísimo partido al chip AY.

El resultado final es que la música no está a un nivel de calidad equiparable al resto de elementos pero, como mal menor, siempre se puede desactivar en el menú.

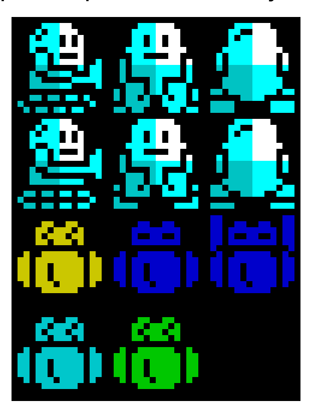
Así que me quedo con la tremenda satisfacción de haber podido participar en la elaboración de un producto de la calidad de Sokoban y con las ganas de sacarme esa espinita clavada en una futura ocasión.

Por último, Javier Vispe nos relata...

CÓMO HACER LA VIDA IMPOSIBLE AL PROGRAMADOR

Con esta frase se puede resumir todo mi trabajo en la parte gráfica de Sokoban. Supongo que Miguel se habrá preguntado durante estos meses por qué se le ocurrió enseñarme el juego que estaba haciendo. No hay cosa peor que mostrar tus proyectos a alguien que no sabe programar, componer música o diseñar niveles. ¡Todos saben que se ofrecerá para diseñar los gráficos!

Mi relación con Sokoban empezó cuando vi las primeras capturas del juego. Ya estaban hechas las cajas, las metas y los bloques que se usan para dibujar los laberintos, pero faltaba el protagonista. Yo, que soy muy majo, sin decir nada, me puse a garabatear en el programa de retoque de imagen, a ver que me salía. La idea del robot surgió rápidamente. Siempre se nos ha vendido la idea de que son herramientas ideales para hacer tareas pesadas, como por ejemplo, empujar cajas. Miguel estaba peleándose con los píxeles de un señor visto desde arriba. Le enseñé a JCN-7000, y al final, ocurrió lo de siempre: la máquina le quitó el puesto de trabajo al hombre.

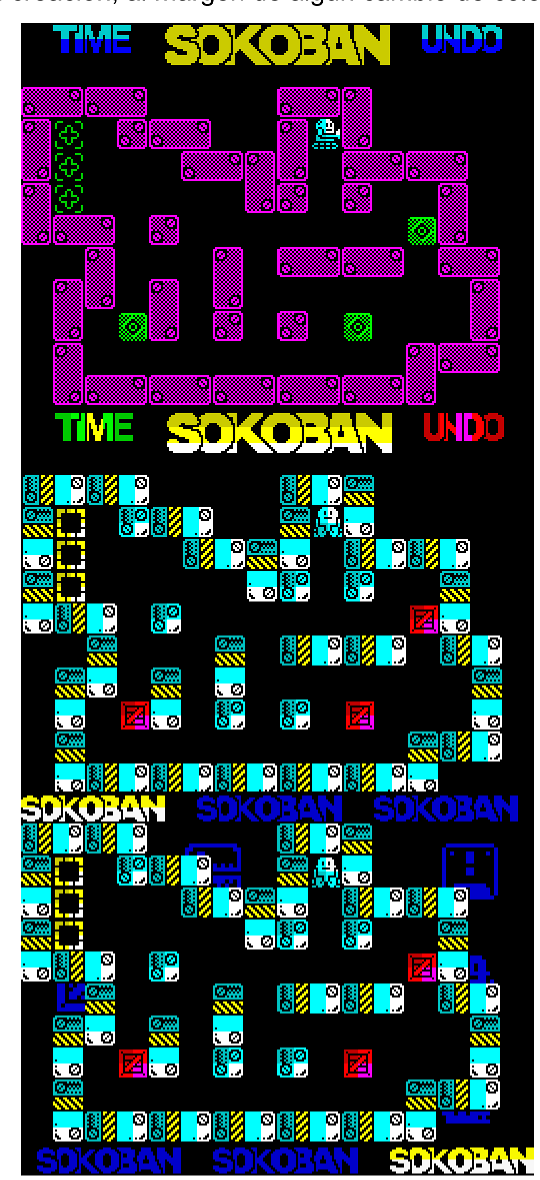


Tras esta primera experiencia, a Miguel se le ocurrió la infeliz idea de que yo podría hacer los "dibujitos" y él dedicarse en cuerpo y alma a la parte del código. Craso error. Al día siguiente ya le había dado la vuelta al set gráfico original a petición suya. Pero es que después de uno, vino otro. Y otro... Y otro... Así fue como me desaté, y me iba comiendo la preciada memoria del Spectrum con ideas cada vez más estrambóticas. Mientras, el señor programador sudaba tinta china para recalcular el mapa de memoria. Aunque, en el fondo, sé que le gusta...



Para hacer los gráficos de Sokoban básicamente he tenido como herramienta de trabajo el PC. Miguel ya había delimitado el tamaño de los tiles, por lo que simplemente tenía que pensar en una temática y plasmarla sobre la pantalla. A día de hoy es un método mucho más cómodo, limpio y rápido. En anteriores ocasiones sí que había trabajado sobre papel, pero en este caso sólo lo he usado para realizar un boceto muy general de la pantalla de carga. Una vez acabados los gráficos, se convertían a ensamblador Z80 con el SevenuP de Metalbrain para insertarlos en el código del juego. En este aspecto, quiero agradecer públicamente que haya compartido su magnífica herramienta, puesto que facilita mucho el trabajo a los desarrolladores.

Tengo que decir que me ha sorprendido lo fácil que me resultó el diseño de los sets gráficos, y que prácticamente no han recibido retoques tras su creación, al margen de algún cambio de color.



En Sokoban hemos utilizado gran cantidad de color en decorados, objetos y personajes. Para evitar la mezcla de atributos se implementó un movimiento de 8 en 8 píxeles, procurando que la respuesta al teclado fuera muy precisa. Un juego puzzle como este permite tomar esta libertad creativa que, por ejemplo, un juego de plataformas no deja. En muchos de estos se ha de contar con un movimiento preciso que no arruine la jugabilidad. En muchas ocasiones eso no se ha tomado en cuenta en Spectrum.

La pantalla de juego fue variando progresivamente conforme se sugerían mejoras. La desaparición de marcadores hizo que nos planteáramos el añadir un fondo que evitara la sensación de una pantalla demasiado vacía con los niveles pequeños. La trama de tiles aumentados, junto al logo del juego cumplía perfectamente esta misión, y no apartaba nuestra atención de la zona de juego. Igualmente, decidimos darle un toque más atractivo a la pantalla de opciones dentro de los niveles.

Uno de los últimos añadidos que hicimos fue la animación del robot apareciendo al principio de cada fase. La buena organización y optimización de código que hizo Miguel nos permitió contar con memoria de sobra para añadir un total de 27 fotogramas para dar el toque final al acabado del juego.

Sólo me queda hablar de lo mucho que he disfrutado del tiempo que he invertido en ayudar a hacer realidad el sueño de Miguel. Gracias a él, también se han cumplido los de un chaval que en tiempos fantaseaba con hacer juegos en código máquina para Spectrum.

¿SE LO ENVUELVO PARA REGALO?

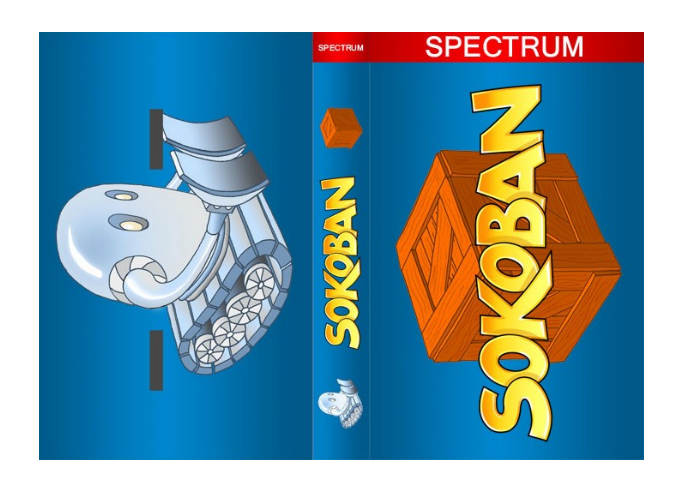
Una de las señas de identidad de los juegos de los 80 era la portada. Muchos de ellos vendieron gracias a la ilustración que incorporaban en el frontal de la caja. Cuantas decepciones han ocultado exhuberantes mujeres, lejanos paisajes espaciales o parajes remotos de nuestro planeta...

Para la realización de la carátula de Sokoban, contamos con Juanje Gómez, nuestro ilustrador favorito de portadas en Magazine ZX. Sin dejarlo recuperarse del jet lag que traía de su viaje a Australia, le endilgamos un nuevo incordio en su buzón de correo.

Como podéis ver, el primer diseño optaba por una disposición horizontal, que se nos hacía extraña a la clásica vertical. De nuevo, entró en acción mi espíritu incordiador y empecé a jugar con el material en el programa de retoque de imagen. Me incliné por una distribución más tradicional, incluyendo una pequeña sinopsis y capturas de pantalla en la parte trasera. También le di un poco de alegría al fondo con una trama de tiles del juego.

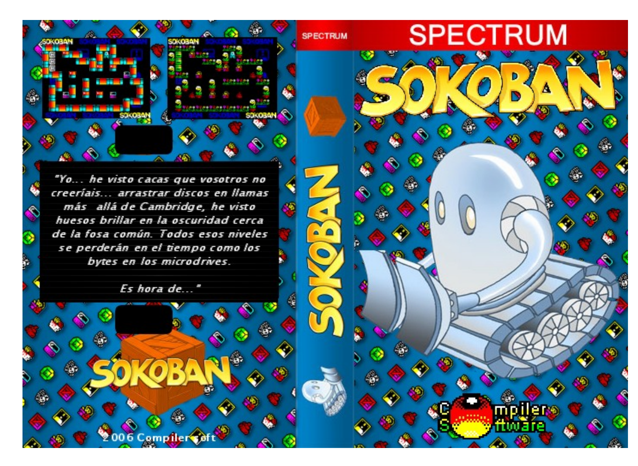
Tras varias pruebas, le envié un boceto a Juanje

para conocer su opinión y ver si podía servirle de algo para retocar la portada original. No sé si lo dijo por quitarse al "pesado" de encima ;), pero le gustó y decidió adoptar el diseño para realizar la magnífica versión final que podéis apreciar. De nuevo, y aún a riesgo de coger fama de empalagoso, tengo que agradecer a Juanje que siempre esté ahí para colaborar en nuestras locuras.

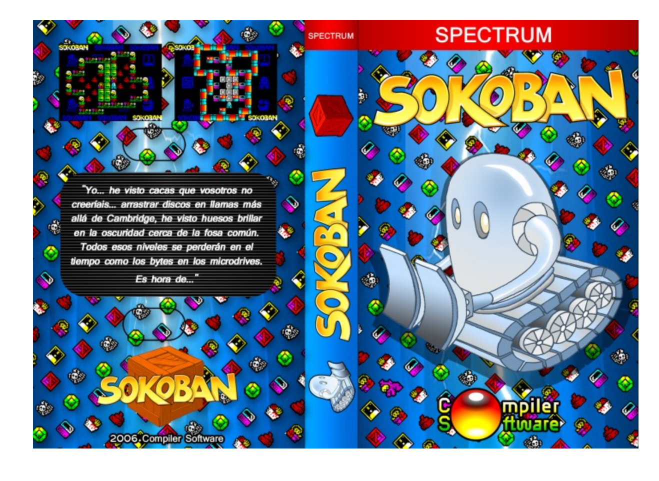






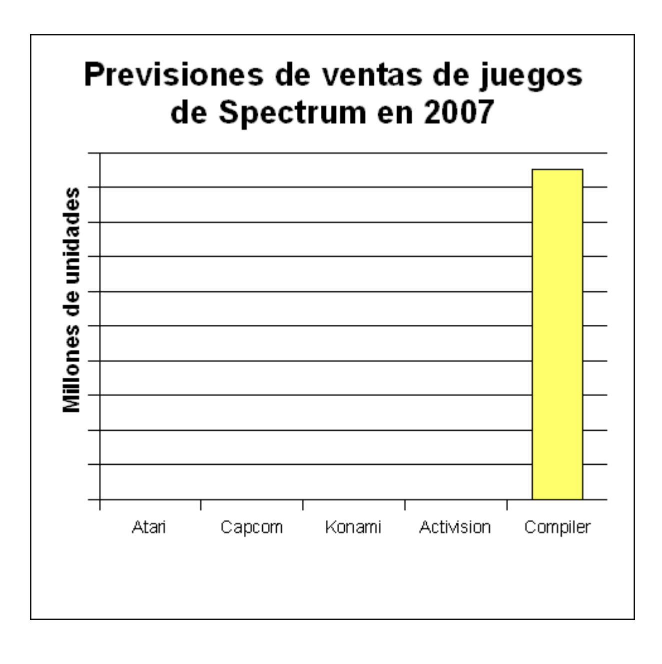






...Y SOKOBAN SE HIZO CINTA

Compiler soft se ha propuesto como meta convertirse en líder del mercado de los 8 bits, superando a grandes empresas con una larga tradición de lanzamientos, como Atari, Capcom, Konami o Activision. Por ello, ha producido versiones físicas de Sokoban para que formen parte de las colecciones de los aficionados al Spectrum. Las previsiones de ventas anuales son muy halagüeñas, como se puede ver en la siguiente gráfica:



Hemos realizado dos ediciones en diferente soporte: cinta y disco de tres pulgadas. El primero se ha acompañado de un póster para las primeras unidades, mientras que el segundo es una edición limitada destinada en exclusiva a los colaboradores del proyecto. Ambas cuentan con las versiones española e internacional del juego.

El proceso de fabricación se ha dividido en dos partes, la carátula y la grabación en cinta. Para el primero, encargamos a una empresa de servicios de impresión digital la portada con las instrucciones a todo color, con sus respectivos pliegues. Para el segundo, estuvimos buscando lugares donde todavía duplicaran casetes. En nuestro país empieza a ser complicado hallar lugares que acepten tiradas pequeñas. A esta dificultad hay que añadir la posibilidad de que el proveedor tenga material viejo, y se encuentre en malas condiciones.

Finalmente encargamos el trabajo a una empresa inglesa. Tras suministrar el máster, nos enviaron una cinta de prueba para comprobar si la grabación era adecuada para la carga en un Spectrum. Una vez certificado que no había problemas, se dio luz verde a la duplicación.

Tras recibir todos los pedidos, sólo quedaba montarlos y prepararlos para su venta. Tras casi un año de desarrollo constante, Sokoban por fin se había materializado en formato físico. Esta última fase ha supuesto un gran esfuerzo, no sólo económico, sino también de tiempo buscando proveedores que ofrecieran lo que necesitábamos y solventando los problemas que han surgido durante la fabricación. Por suerte, la ilusión de ver nuestro sueño cumplido, es una satisfacción que no olvidaremos fácilmente.

Ahora, tendremos que ir pensando qué vamos a hacer en los próximos meses con tanto tiempo libre...

COMPILER SOFTWARE



programación ensamblador

LENGUAJE ENSAMBLADOR DEL Z80 (II)

DESPLAZAMIENTO DE MEMORIA, MANIPULACIÓN DE BITS Y OPERACIONES LÓGICAS

En nuestra anterior entrega comenzamos nuestra andadura en el lenguaje ensamblador del Z80 por medio de las instrucciones de carga (LD), operaciones aritméticas (ADD, ADC, SUB, SBC, INC, DEC) y de intercambio (EXX y EX). Mientras se introducían las diferentes instrucciones, os mostramos la manera de emplear los registros y cómo los resultados podían afectar a los flags del registro F, mediante las "tablas de afectación de flags".

Toda la teoría explicada en el anterior capítulo del curso nos permitirá avanzar ahora mucho más rápido, ya que con todos los conceptos asimilados podemos ir realizando una rápida introducción a nuevas instrucciones, bastando ahora con una simple descripción de cada una de ellas. Las tablas de afectación de flags y comentarios sobre los operandos permitidos (o prohibidos) para cada una de ellas completarán la formación necesaria.

Para poder continuar con éste y posteriores capítulos del curso será imprescindible haber comprendido y asimilado todos los conocimientos de las entregas anteriores, de modo que si no es así, recomendamos al lector que relea las entregas 1, 2 y 3, y que se asegure de comprender todos los conceptos explicados.

En esta entrega trataremos las operaciones con bits (NEG, CPL, BIT, SET y RES), las operaciones lógicas (AND, OR y XOR) y las operaciones de desplazamiento de bits (RR, RL, RLC, RRC, SLA, SRA y SRL).

No obstante, antes de pasar a hablar de las operaciones con bits finalizaremos con la descripción de las instrucciones de carga (en este caso las repetitivas), y veremos 4 instrucciones muy sencillas: SCF, CCF, NOP y DAA.

INSTRUCCIONES DE DESPLAZAMIENTO DE MEMORIA

En la entrega anterior conocimos la existencia de las instrucciones de carga (LD), que nos permitían mover valores entre registros. Lo que vamos a ver a continuación es cómo podemos copiar un byte de una posición de memoria a otra, con una sóla instrucción.

Las 2 instrucciones que vamos a describir: LDI y LDD, no admiten parámetros. Lo que hacen estas instrucciones es:

LDI (Load And Increment):

- Leer el byte de la posición de memoria apuntada por el registro HL.
- Escribir ese byte en la posición de memoria apuntada por el registro DE.
- Incrementar DE en una unidad (DE=DE+1).
- Incrementar HL en una unidad (HL=HL+1).
- Decrementar BC en una unidad (BC=BC-1).

LDD (Load And Decrement):

- Leer el byte de la posición de memoria apuntada por el registro HL.
- Escribir ese byte en la posición de memoria apuntada por el registro DE.
- Decrementar DE en una unidad (DE=DE-1).
- Decrementar HL en una unidad (HL=HL-1).
- Decrementar BC en una unidad (BC=BC-1).

En pseudocódigo:

```
LDI:     Copiar [HL] en [DE]
         DE=DE+1
         HL=HL+1
         BC=BC-1
```

```
LDI:    Copiar [HL] en [DE]
        DE=DE+1
        HL=HL+1
        BC=BC-1

LDD:    Copiar [HL] en [DE]
        DE=DE-1
        HL=HL-1
        BC=BC-1
```

Estas instrucciones lo que nos permiten es copiar datos de una zona de la memoria a otra. Por ejemplo, supongamos que queremos copiar el byte contenido en 16384 a la posición de memoria 40000:

Qué tiene de especial LDI con respecto a realizar la copia a mano con operaciones LD? Pues que al incrementar HL y DE, lo que hace es apuntar a los siguientes elementos en memoria (HL=16385 y DE=40001), con lo cual nos facilita la posibilidad de copiar múltiples datos (no sólo 1), con varios LDI. Lo mismo ocurre con LDD, que al decrementar DE y HL los hace apuntar a los bytes anteriores de origen y destino.

```
LD HL, 16384
 LD DE, 40000
 LDI
```

Pero para facilitarnos más aún la tarea de copia (y no tener que realizar bucles manualmente), el Z80 nos proporciona las instrucciones LDIR y LDDR, que funcionan igual que LDI y LDD pero copiando tantos bytes como valga el registro BC. Es decir:

```
LDIR = Repetir LDI hasta que BC valga 0
     = Repetir:
          Copiar [HL] en [DE]
          DE=DE+1
          HL=HL+1
          BC=BC-1
       Hasta que BC = 0

LDDR = Repetir LDD hasta que BC valga 0
     = Repetir:
          Copiar [HL] en [DE]
          DE=DE-1
          HL=HL-1
          BC=BC-1
       Hasta que BC = 0
```

Estas instrucciones son enormemente útiles porque nos permiten copiar bloques de datos desde una zona de la memoria a otra. Por ejemplo, podemos hacernos una copia del estado de la pantalla en una zona de memoria mediante:

```
 LD HL, 16384
LD DE, 50000
LD BC, 6912
LDIR
```

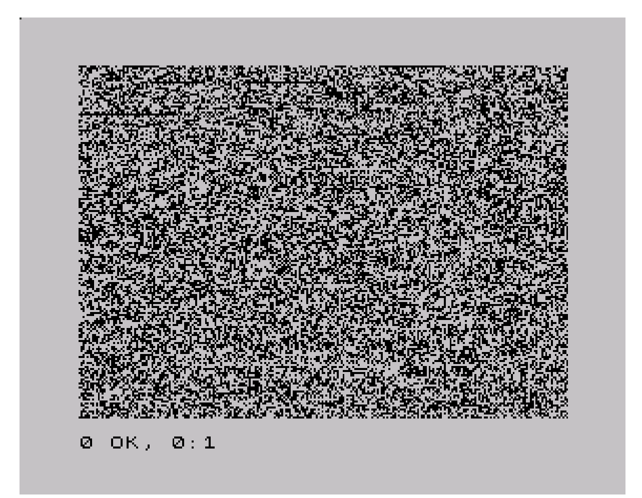
Con el anterior programa, copiamos los 6912 bytes que hay a partir de la dirección de memoria 16384 (la pantalla) y los copiamos a partir de la dirección 50000. De este modo, desde 50000 a 56912 tendremos una copia del estado de la pantalla (podría servir, por ejemplo, para modificar cosas en esta "pantalla virtual" y después copiarla de nuevo a la videoram, tomando HL=50000 y DE=16384).

Para demostrar esto, ensamblemos y ejecutemos el siguiente ejemplo:

```
; Ejemplo de LDIR donde copiamos 6144 bytes de la ROM
; a la videomemoria. Digamos que "veremos la ROM" :)
ORG 40000

 LD HL, 0          ; Origen: la ROM
 LD DE, 16384      ; Destino: la VideoRAM
 LD BC, 6144       ; toda la pantalla
 LDIR              ; copiar

RET
```



Aspecto de la rom al copiarla a la VRAM

Os animo a que probéis el equivalente BASIC del ejemplo anterior y verifiquéis las diferencias de velocidad existentes:

```
 5 REM Ejecutar "bas2tap -a10
   copiarom.bas copiarom.tap"
10 REM Copiamos la ROM en la
   VideoRAM
20 FOR I=0 TO 6144 : POKE
   (16384+I), (PEEK I) : NEXT I
30 PAUSE 0
```

Concluimos pues que en todas estas instrucciones de copia de memoria o transferencia, HL es el origen, DE el destino y BC el número de bytes a transferir. Con LDI y LDD sólo copiaremos 1 byte (independientemente del valor de BC, aunque lo decrementará), y con LDIR y LDDR copiaremos tantos bytes como valga BC, decrementando BC hasta que su valor llega a cero. Los flags quedarán afectados, especialmente con LDI y LDD para indicarnos mediante el registro P/V si BC ha llegado a cero.

```
                      Flags
  Instrucción       |S Z H P N C|
----------------------------------
  LDI               |- - 0 * 0 -|
  LDD               |- - 0 * 0 -|
  LDDR              |- - 0 0 0 -|
  LDIR              |- - 0 0 0 -|
```

Recordemos el significado de los símbolos de la tabla de afectación de flags (válido para todas las tablas de instrucciones que utilizaremos a lo largo del curso):

```
- = El flag NO se ve afectado por la operación.
* = El flag se ve afectado por la operación acorde al resultado.
0 = El flag se pone a cero.
1 = El flag se pone a uno.
V = El flag se comporta como un flag de Overflow acorde al resultado.
P = El flag se comporta como un flag de Paridad acorde al resultado.
? = El flag toma un valor indeterminado.
```

Una duda que puede asaltarle al lector es: "si tenemos LDIR para copiar bloques, ¿para qué nos puede servir LDDR? ¿No es una instrucción redundante, que podemos no necesitar nunca gracias a LDIR? Pues como bien nos apunta Miguel (devil_net) LDDR es especialmente útil cuando hay que hacer copias de bloques de datos que se superponen.

Supongamos que tenemos que realizar una copia de 1000 bytes desde 25000 hasta 25100. Supongamos que preparamos el siguiente código:

```
LD HL, 25000
LD DE, 25100
LD BC, 1000
LDIR
```

Este código no funcionará como esperamos. Ambas zonas se superponen, con lo cual si lo ejecutamos, ocurrirá lo siguiente:

- El byte en [25000] se copiará a [25100].
- El byte en [25001] se copiará a [25101].
- etc...

¿Qué ocurrirá cuando LDIR llegue al byte número 25100 y lo intente copiar a 25200? Sencillamente, que hemos perdido el contenido REAL del byte número 25100, porque fue machacado al principio de la ejecución del LDIR por el byte contenido en [25000]. No estamos moviendo el bloque correctamente, porque las zonas se superponen y

cuando llegamos a la zona destino, estamos copiando bytes que movimos desde el origen.

Para ello, lo correcto sería utilizar el siguiente código:

```
LD HL, 26000
LD DE, 26100
LD BC, 1000
LDDR
```

Es decir, apuntamos HL y DE al final de los 2 bloques de copia, y copiamos los bloques desde abajo, decrementando. De este modo nunca "machacamos" una posición de memoria que vayamos a copiar posteriormente con un dato.

En este ejemplo:

- El byte en [26000] se copia en [26100].
- El byte en [25999] se copia en [26099].
- El byte en [25998] se copia en [26098].
- (...)
- El byte en [25001] se copia en [25101].
- El byte en [25000] se copia en [25100].

Que es, efectivamente, lo que queríamos hacer, pero sin perder datos en la copia: copiar 1000 bytes desde 25000 a 25100 (sólo que realizamos la copia de abajo a arriba).

UN EJEMPLO DE RUTINA CON LDIR

Vamos a ver un ejemplo de rutina en ensamblador que utiliza LDIR con un propósito concreto: vamos a cargar una pantalla de carga (por ejemplo, para nuestros juegos) de forma que no aparezca poco a poco como lo haría con LOAD "" SCREEN$, sino que aparezca de golpe.

Para eso lo que haremos será lo siguiente:

Crearemos una rutina en ensamblador que copiará 6912 bytes desde la dirección 50000 hasta la posición 16384 (la videoram). La rutina ya la hemos visto:

```
ORG 40000
 LD HL, 50000     ; Origen: 50000
 LD DE, 16384     ; Destino: la VideoRAM
 LD BC, 6912      ; toda la pantalla
 LDIR             ; copiar
RET
```

La ensamblamos con pasmo y obtenemos el siguiente código máquina:

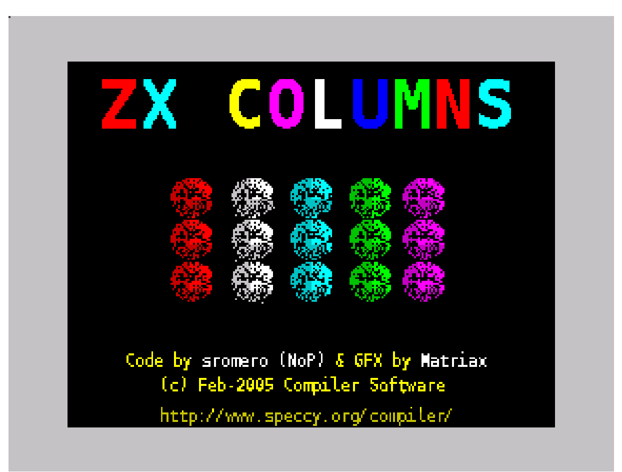
```
 33, 80, 195, 017, 0, 64, 1, 0, 27, 237, 176, 201
```

Nos crearemos un cargador BASIC que realice el trabajo de pokear nuestra rutina en 40000 y cargar la pantalla en 50000:

```
10 REM Ejemplo de volcado de pantalla de carga
20 CLEAR 39999
30 DATA 33, 80, 195, 017, 0, 64, 1, 0, 27, 237, 176, 201
40 FOR I=0 TO 11 : READ OPCODE : POKE 40000+I, OPCODE : NEXT I
50 LOAD "" CODE 50000, 6912
60 RANDOMIZE USR 40000
70 PAUSE 0
```

Grabamos este cargador en cinta (o tap/tzx), y a continuación, tras el cargador, grabamos una pantalla de carga.

Ejecutamos el programa resultante en emulador o Spectrum, y veremos cómo la carga de la pantalla no puede verse en el monitor. Cuando está termina su carga, la rutina ensamblador se ejecuta y se vuelca, de golpe, a la videoram (estad atentos a la carga, porque el volcado es muy rápido).



La pantalla de carga de ZXColumns, volcada a videoram

ALGUNAS INSTRUCCIONES ESPECIALES

Antes de comenzar con las instrucciones de manipulación de registros y datos a nivel de bits vamos a ver una serie de instrucciones difíciles de encuadrar en futuros apartados y que pueden sernos de utilidad en nuestros programas:

- SCF: Set Carry Flag

  Esta instrucción (que no admite parámetros) pone a 1 el Carry Flag del registro F. Puede sernos útil en determinadas operaciones aritméticas.

- CCF: Complement Carry Flag

  Esta instrucción (que tampoco admite parámetros) invierte el estado del bit de Carry Flag: si está a 1 lo pone a 0, y viceversa. Puede servirnos para poner a 0 el carry flag mediante la combinación de SCF + CCF.

- NOP: No OPeration

  Esta instrucción especial del microprocesador ocupa un byte en el código (opcode 00h) y no efectúa ninguna operación ni afecta a ningún flag. Eso sí, se toma 4 t-states (t-estados, o ciclos del procesador) para ejecutarse. ¿Para qué puede servir una instrucción que no realiza ninguna acción y que requiere tiempo del procesador (aunque sea muy poco) para ejecutarse? Muy sencillo: para múltiples cosas. Por un lado, podemos utilizarla en bucles de retardos (varios NOPs ejecutados en un bucle que se repita varias veces) para poner retardos en nuestros programas o juegos. Por otro, como ocupa un byte en memoria (en el código) y no realiza ninguna operación, podemos utilizarla para rellenar zonas de nuestro código, y así alinear código posterior en una determinada dirección que nos interese.

- DAA: Decimal Adjust Accumulator

  Esta instrucción permite realizar ajustes en los resultados de operaciones con números BCD (tras operaciones aritméticas). ¿Qué son los números en formato BCD? Es una manera de representar números en los registros (o memoria) de forma que de los 8 bits de un byte se utilizan los 4 bits del 0 al 3 para representar un número del 0 al 9 (4 bits = desde 0000 hasta 1111), y los 4 bits del bit 4 al 7 para representar otro número del 0 al 9. A los 2 números BCD juntos se les llama "Byte BCD" o "números en formato BCD". Un número BCD puede estar formado por varios bytes BCD, siendo cada byte 2 cifras del mismo. Así, para representar un número de 10 cifras en BCD sólo es necesario utilizar 5 bytes. Además, podemos utilizar un byte extra que indique la posición de la "coma decimal" para así poder trabajar con números decimales en ensamblador. Si queremos realizar operaciones entre este tipo de números deberemos programarnos nosotros mismos las rutinas para realizarlas.

A lo largo del curso no utilizaremos números en BCD y por lo tanto es muy probable que no lleguemos a utilizar DAA, pero conviene saber que el Z80 nos brinda la oportunidad de utilizar números más grandes de 16 bits, operando con números en BCD. Para realizar juegos normalmente no necesitaremos de estas instrucciones.

Todas estas instrucciones afectan a los flags de la siguiente manera:

```
                        Flags
 Instrucción         |S Z H P N C|
---------------------------------
 SCF                 |- - 0 - 0 1|
 CCF                 |- - ? - 0 *|
 NOP                 |- - - - - -|
 DAA                 |* * * P - *|
```

OPERACIONES CON BITS

El conjunto de instrucciones que vamos a ver hoy está pensado para trabajar con los bits individuales de un registro: invertir los bits de un registro, obtener el complemento a dos de un registro y poner a 0 o a 1, o probar, un determinado bit de un registro.

CPL Y NEG

CPL es una instrucción que se usa para obtener el inverso del registro A. No admite parámetros (el operando destino es el registro A) y cuando la ejecutamos, se invierte el estado de cada uno de los bits de A, de forma que los unos pasan a valer cero, y los ceros, uno.

```
LD A, %10000001
CPL                  ; A = %01111110
```

La tabla de afectación de flags de CPL es:

```
                           Flags
  Instrucción       |S Z H P N C|
----------------------------------
  CPL               |- - 1 - 1 -|
```

Es decir, se deja a uno el flag de Resta (N) y el de HalfCarry (H). El resto de flags no se ven afectados.

Existe una instrucción similar a CPL, pero que además de realizar la inversión de unos y ceros suma 00000001 al resultado de la inversión del registro A. El resultado es que en A obtenemos el valor negativo del número en complemento a dos almacenado en este registro (A = -A).

Por ejemplo:

```
LD A, 1      ; A = +1
NEG          ; A = -1 = %11111111
```

La tabla de afectación de flags de NEG es:

```
                           Flags
  Instrucción       |S Z H P N C|
----------------------------------
  NEG               |* * * V 1 *|
```

SET, RES Y BIT

Las siguientes instrucciones que vamos a ver nos permitirán el manejo de cualquiera de los bits de un registro o posición de memoria: activar un bit (ponerlo a uno), desactivar un bit (ponerlo a cero), o testear su valor (averiguar si es cero o uno) afectando a los flags.

Comencemos con "SET". Esta instrucción activa (pone a valor 1) uno de los bits de un registro o dirección de memoria. El formato de la instrucción es:

```
SET bit, DESTINO
```

donde Bit es un número entre 0 (el bit menos significativo o bit 0) y 7 (el de más valor o más significativo), y destino puede ser cualquier registro de 8 bits (A, B, C, D, E, H y L), una dirección de memoria apuntada por HL (es decir, el destino puede ser [HL]), o una dirección de memoria indexada por [IX+N] o [IY+N]. Con esto, las siguientes instrucciones serían válidas:

```
SET 5, A          ; Activar el bit 5 del registro A
SET 0, H          ; Activar el bit 0 del registro H
SET 7, [HL]       ; Activar el bit 7 del dato contenido en
                  ; la dirección de memoria apuntada por HL
SET 1, [IX+10]    ; Activar el bit 1 del dato en [IX+10]
```

La instrucción contraria a SET es RES (de reset), que pone a cero uno de los bits del destino especificado. Su formato es igual que el de SET, como podemos ver en los siguientes ejemplos:

```
RES bit, DESTINO

RES 0, H          ; Desactivar el bit 0 del registro H
RES 7, [HL]       ; Desactivar el bit 7 del dato contenido en
                  ; la dirección de memoria apuntada por HL
RES 1, [IX-5]     ; Desactivar el bit 0 del dato en [IX-5]
```

SET y RES no afectan a los flags, como podemos ver en su tabla de afectación de indicadores:

```
                        Flags
 Instrucción       |S Z H P N C|
----------------------------------
 SET b, s          |- - - - - -|
 RES b, s          |- - - - - -|
```

La última instrucción de manipulación de bits individuales que veremos en este apartado es BIT. Esta instrucción modifica el flag de cero (Z) y deja su valor en 0 ó 1 dependiendo del valor del bit que estamos probando. Si estamos comprobando, por ejemplo, el bit 5 del registro A, ocurrirá lo siguiente:

- Si el bit 5 del registro A es cero: el Flag Z se pone a 1.
- Si el bit 5 del registro A es uno: el flag Z se pone a 0.

En otras palabras, Z toma la inversa del valor del Bit que comprobamos: esto es así porque Z no es una COPIA del bit que estamos comprobando, sino el resultado de evaluar si dicho bit es cero o no, y una evaluación así pone a uno el flag Z sólo cuando lo que se evalúa es cero.

Su formato es:

```
BIT bit, DESTINO
```

El destino puede ser el mismo que en SET y RES: un registro, posición de memoria apuntado por HL o posición de memoria apuntada por un registro índice más un desplazamiento.

Por ejemplo:

```
LD A, 8        ; A = %00001000
BIT 7, A       ; El flag Z vale 1
               ; porque el bit 7 es 0
BIT 3, A       ; El flag Z vale 0
               ; porque el bit 3 no es 0
               ; (es 1).
```

El lector se preguntará ... ¿cuál es la utilidad de BIT? Bien, el hecho de que BIT modifique el Zero Flag de acuerdo al bit que queremos comprobar nos permitirá utilizar instrucciones condicionales para realizar muchas tareas. Por ejemplo, podemos comprobar el bit 0 de un registro (algo que nos permitiría saber si es par o impar) y en caso de que se active el flag de Zero (Si z=1, el bit 0 vale 0, luego es par), realizar un salto a una determinada línea de programa.

Por ejemplo:

```
BIT 0, A       ; ¿Que valor tiene el bit 0?
               ; Ahora Z = bit 0 de A.
JP Z es_par    ; Saltar si esta Z activado
               ; (si Z=1 -> salta a es_par)
               ; ya que si Z=1, es porque e
```

Esta instrucción es, como veremos en muchas ocasiones, muy útil, y como ya hemos dicho sí que altera el registro F:

```
                        Flags
 Instrucción       |S Z H P N C|
----------------------------------
 BIT b, s          |? * 1 ? 0 -|
```

ROTACION DE BITS

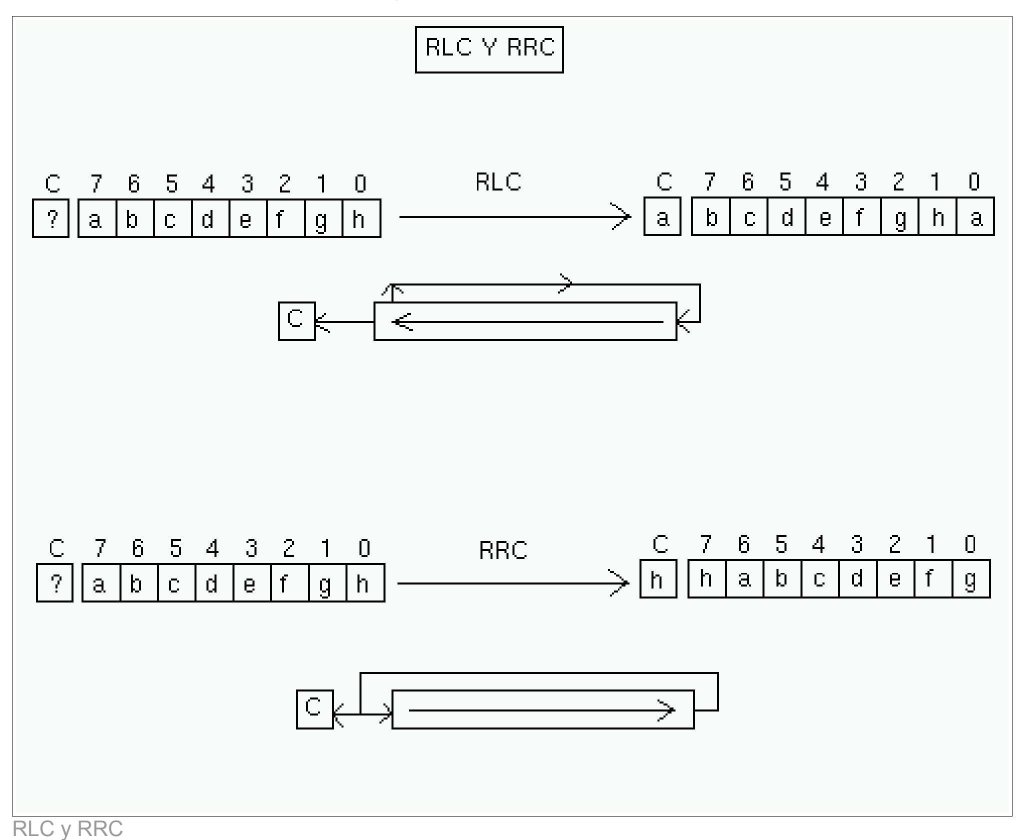
El siguiente set de instrucciones que veremos nos permitirá ROTAR (ROTATE) los bits de un dato de 8 bits (por ejemplo, almacenado en un registro o en memoria) hacia la izquierda o hacia la derecha.

Para realizar esta tarea tenemos disponibles 2 instrucciones básicas: RLC y RRC. La primera de ellas, RLC, rota el registro o dato en un bit a la izquierda (RLC = Rotate Left Circular), y la segunda lo hace a la derecha.

```
Bit      7 6 5 4 3 2 1 0                 7 6 5 4 3 2 1 0
       ----------------- -> RLC ->     -----------------
Valor    a b c d e f g h                 b c d e f g h a


Bit      7 6 5 4 3 2 1 0                 7 6 5 4 3 2 1 0
       ----------------- -> RRC ->     -----------------
Valor    a b c d e f g h                 h a b c d e f g
```

Así, RLC de 00000001 daría como resultado 00000010. Como la rotación es circular, todos los bits se mueven una posición a la izquierda y el bit 7 se copia en el bit 0. Así mismo, RRC de 00000001 daría como resultado 10000000, ya que el bit 0 al rotarse a la derecha (como todos los demás bits) se copia donde estaba el bit 7. Cabe destacar que el Carry Flag se vé afectado, ya que el bit 7 en RLC y el 0 en RRC también se copiará allí.



RLC y RRC

Por ejemplo, supongamos el valor 10000001 almacenado en el registro B:

El resultado las 2 operaciones descritas sería:

```
LD B, %10000001     ; B = 10000001
RLC B               ; B = 00000011

LD B, %10000001     ; B = 10000001
RRC B               ; B = 00000011
```

No sólo podemos rotar registros: en general el destino de la rotación podrá ser un registro, el contenido de la dirección de memoria apuntada por [HL], o bien el contenido de la memoria apuntada por un registro índice más desplazamiento ([IX+N] o [IY+N]). Más adelante

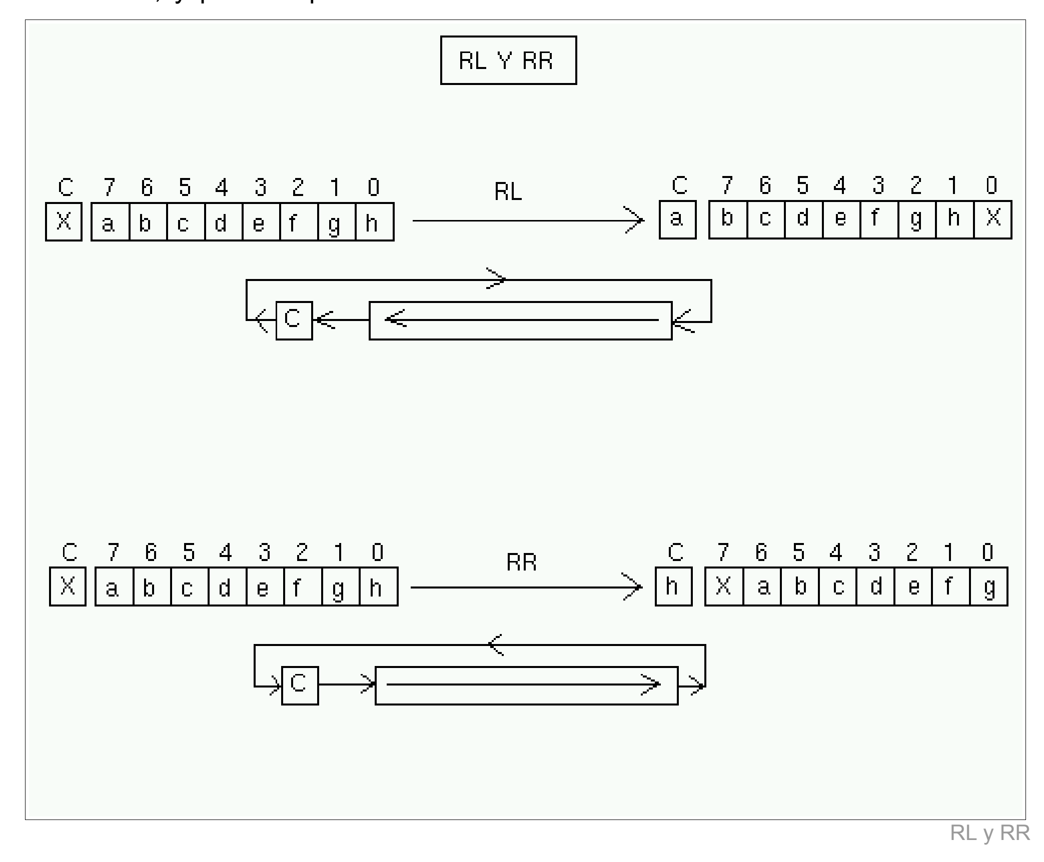
veremos la tabla de afectación de flags de esta y otras instrucciones que veremos a continuación.

Además de RLC y RRC (rotación circular), tenemos disponibles 2 instrucciones más que nos permiten apoyarnos en el Carry Flag del registro F como si fuera un bit más de nuestro registro, comportándose como el noveno bit (de más valor) del registro: hablamos de las instrucciones RL y RC:

```
Bit      C    7 6 5 4 3 2 1 0               C    7 6 5 4 3 2 1 0
        ----------------------- ->  RL  ->  ---------------------
Valor    X    a b c d e f g h               a    b c d e f g h X


Bit      C    7 6 5 4 3 2 1 0               C    7 6 5 4 3 2 1 0
        ----------------------- ->  RR  ->  ---------------------
Valor    X    a b c d e f g h               h    X a b c d e f g
```

El CarryFlag hace de bit extra: por un lado se copia al Bit 0 o al Bit 7 según estemos rotando a izquierda o a derecha, y por otra parte recibe el valor del bit 7 del bit 0 (respectivamente para RL y RR).



RL y RR

Por ejemplo, supongamos el valor 10000001 almacenado en el registro B y que el carry flag estuviera a uno:

El resultado las 2 operaciones descritas sería:

```
SCF                 ; Set Carry Flag (hace C=1)
LD B, %00000010     ; B = 00000010
RL B                ; B = 00000101 y C=0 (del bit 7)

SCF                 ; Set Carry Flag (hace C=1)
LD B, %01000001     ; B = 01000000
RR B                ; B = 10100000 y C=1 (del bit 0)
```

Así pues, RLC y RRC son circulares y no utilizan el Carry Flag, mientras que RR y RL sí que lo utilizan, como un bit extra. Utilizando RR/RL 9 veces o bien RLC/RRC 8 veces sobre un mismo registro obtenemos el valor original antes de comenzar a rotar.

Veamos la tabla de afectación de flags de estas

nuevas instrucciones:

```
                       Flags
 Instrucción       |S Z H P N C|         Significado
-----------------------------------------------------------------
 RLC s             |* * 0 P 0 *|         Rotate Left Circular
 RRC s             |* * 0 P 0 *|         Rotate Right Circular
 RL s              |* * 0 P 0 *|         Rotate Left (con Carry)
 RR s              |* * 0 P 0 *|         Rotate Right (con Carry)
```

El destino "s" puede ser cualquier registro de 8 bits, memoria apuntada por HL o registros índice con desplazamiento. Como veis hay muchos flags afectados, y en esta ocasión el flag P/V ya no nos sirve para indicar desbordamientos sino que su estado nos da la PARIDAD del resultado de la operación de rotación. Con el flag P a uno, tenemos paridad par (even), es decir, el número de bits a uno en el resultado es par. Si está a cero significa que el número de bits a uno en el resultado es impar.

Aunque pueda parecer sorprendente (ya que podemos utilizar las 4 operaciones anteriores con el registro A como operando), existen 4 instrucciones más dedicadas exclusivamente a trabajar con "A": hablamos de RLA, RRA, RLCA y RRCA. La diferencia entre estas 4 instrucciones y su versión con un espacio en medio (RL A, RR A, RLC A y RRC A) radica simplemente en que las nuevas 4 instrucciones (sin espacio) alteran los flags de una forma diferente:

```
                       Flags
 Instrucción       |S Z H P N C|       Significado
-----------------------------------------------------------------
 RLA               |- - 0 - 0 *|       Rotate Left Accumulator
 RRA               |- - 0 - 0 *|       Rotate Right Accumulator
 RLCA              |- - 0 - 0 *|       Rotate Left Circular Acc.
 RRCA              |- - 0 - 0 *|       Rotate Right Circular Acc.
```

Como veis, están pensadas para alterar MENOS flags que sus homónimas de propósito general (algo que nos puede interesar en alguna ocasión).

Y para acabar con las instrucciones de rotación, tenemos RLD y RRD, que realiza una rotación entre A y el contenido de la memoria apuntada por HL.

Concretamente, RRD lo que hace es:

- Leer el dato contenido en la dirección de memoria apuntada por HL.
- Coger los 4 bits más significativos (bit 4-7) de ese valor.
- Rotar A hacia la izquierda 4 veces (copiando los bits 0-3 en las posiciones 4-7).
- Copiar los 4 bits extraídos de la memoria en los 4 bits menos significativos de A.

Resumiendo, supongamos los siguientes valores de A y [HL]:

```
Registro A:   Bit 7 6 5 4 3 2 1 0
              --------------------
                  a b c d e f g h

[HL]:         Bit 7 6 5 4 3 2 1 0
              --------------------
                  s t u v w x y z
```

Resultado de RRD:

```
Registro A:   Bit 7 6 5 4 3 2 1 0
              --------------------
                  e f g h s t u v
```

Resultado de RLD:

```
Registro A:   Bit 7 6 5 4 3 2 1 0
              --------------------
                  s t u v e f g h
```

En pseudocódigo C:

```
RRD:  A = ( A<<4 )     | ([HL]>>4)
RLD:  A = ( [HL]<<4 ) | (A & 0x0F)
```

La afectación de flags sería:

```
                      Flags
 Instrucción        |S Z H P N C|        Significado
-----------------------------------------------------------------
 RLD                |* * 0 P 0 -|        Rotate Left 4 bits
 RRD                |* * 0 P 0 -|        Rotate Right 4 bits
```

Aunque ahora todo este conjunto de instrucciones pueda parecernos carente de utilidad, lo que hace el microprocesador Z80 es proveernos de toda una serie de pequeñas herramientas (como estas de manipulación, chequeo y rotación de bits) para que con ellas podamos resolver cualquier problema, mediante la combinación de las mismas. Os aseguramos que en más de una rutina tendréis que usar instrucciones de rotación o desplazamiento.

DESPLAZAMIENTO DE BITS

El siguiente set de instrucciones que veremos nos permitirá DESPLAZAR (SHIFT) los bits de un dato de 8 bits (por ejemplo, almacenado en un registro o en memoria) hacia la izquierda o hacia la derecha. Desplazar es parecido a rotar, sólo que el desplazamiento no es circular; es decir, los bits que salen por un lado no entran por otro, sino que entran ceros:

```
00010001 ROTADO A LA IZQUIERDA es 00100010
(movemos todos los bits hacia la izquierda y el bit 0 entra como 0. El
  bit 7 se copia al Carry)

10000001 ROTADO A LA DERECHA es 01000000
(el 0 del bit 7 del resultado entra nuevo, el 1 del bit 0 origen se
 pierde)
```

Las instrucciones de desplazamiento a izquierda y derecha en Z80 se llaman SLA (Shift Left Arithmetic) y SRA (Shift Right Arithmetic), y su formato es:

```
SRA operando
SLA operando
```

Donde operando puede ser el mismo tipo de operando que en las instrucciones de rotación: un registro de 8 bits, [HL] o [IX/IY+N]. Lo que realizan estas operaciones sobre el dato operando es:

```
Bit 7 6 5 4 3 2 1 0                C    7 6 5 4 3 2 1 0
   ----------------- -> SLA ->  ------------------------
    a b c d e f g h                a    b c d e f g h 0
```

Literalmente:

- Rotar los bits a la izquierda (<<).
- El bit "a" (bit 7) se copia al Carry Flag.
- Por la derecha entra un cero.

```
Bit 7 6 5 4 3 2 1 0                C    7 6 5 4 3 2 1 0
   ----------------- -> SRA ->  ------------------------
    a b c d e f g h                h    a a b c d e f g
```

Literalmente:

- Rotar los bits a la derecha (>>).
- El bit "h" (bit 0) se copia al Carry Flag.
- En la izquierda (bit 7) se mantiene su valor anterior.

Nótese pues que SLA y SRA nos permiten trabajar también con números negativos. En el caso de SLA se utiliza el carry flag para almacenar el estado del bit 7 tras la rotación (con lo cual podemos conservar el signo si sabemos dónde buscarlo). En el caso de SRA, porque el bit 7 además de desplazarse hacia la derecha se mantiene en su posición (manteniendo el signo).

El hecho de desplazar un número binario una posición a izquierda o derecha tiene una curiosa propiedad: el número resultante se multiplica o divide por 2.

Pensemos un poco en nuestro sistema decimal: si tenemos un determinado número y desplazamos todos los dígitos una posición a la izquierda y añadimos un cero, lo que está sucediendo es que multiplicamos el valor del número por la base:

```
  1 5  -> Desplazar y cero  ->  1 5 0
(equivale a multiplicar por la base, es decir, por 10)
```

Si desplazamos el número a la derecha, por contra, estamos dividiendo por la base:

```
  1 5 2 -> Desplazar y cero -> 0 1 5
(equivale a dividir por la base, es decir, por 10).
```

En binario ocurre lo mismo: al desplazar un byte a la izquierda estamos multiplicando por 2, y al hacerlo a la derecha estamos dividiendo por 2 (siempre divisiones enteras). Veamos unos ejemplos:

```
 33 = 00100001
          << 1   (<< significa desplazamiento de bits a izquierda)
    ------------
      01000010 = 66 (33*2)


 14 = 00001110
          >> 1   (>> significa desplazamiento de bits a derecha)
    ------------
      00000111 = 7 (14/2)
```

Cada vez que realizamos un desplazamiento estamos multiplicando el resultado por dos, de forma que:

```
  Dirección        Núm. desplazamientos   Operación
 -----------------------------------------------------------------
  SLA:
  Izquierda (<<)         1                 N = N*2
  Izquierda (<<)         2                 N = (N*2)*2 = N*4
  Izquierda (<<)         3                 N = ((N*2)*2)*2 = N*8
  Izquierda (<<)         4                 N = (...) N*16
  Izquierda (<<)         5                 N = (...) N*32
  Izquierda (<<)         6                 N = (...) N*64
  Izquierda (<<)         7                 N = (...) N*128
  SRA:
  Derecha   (>>)         1                 N = N/2
  Derecha   (>>)         2                 N = (N/2)/2 = N/4
  Derecha   (>>)         3                 N = ((N/2)/2)/2 = N/8
  Derecha   (>>)         4                 N = (...) N/16
  Derecha   (>>)         5                 N = (...) N/32
  Derecha   (>>)         6                 N = (...) N/64
  Derecha   (>>)         7                 N = (...) N/128
```

Así, desplazar una vez a la izquierda equivale a multiplicar por 2, 2 veces, por 4, 3 veces, por 8, etc. En resumen, desplazar un registro N veces a la izquierda equivale a multiplicarlo por 2 elevado a

N. Lo mismo ocurre con la multiplicación.

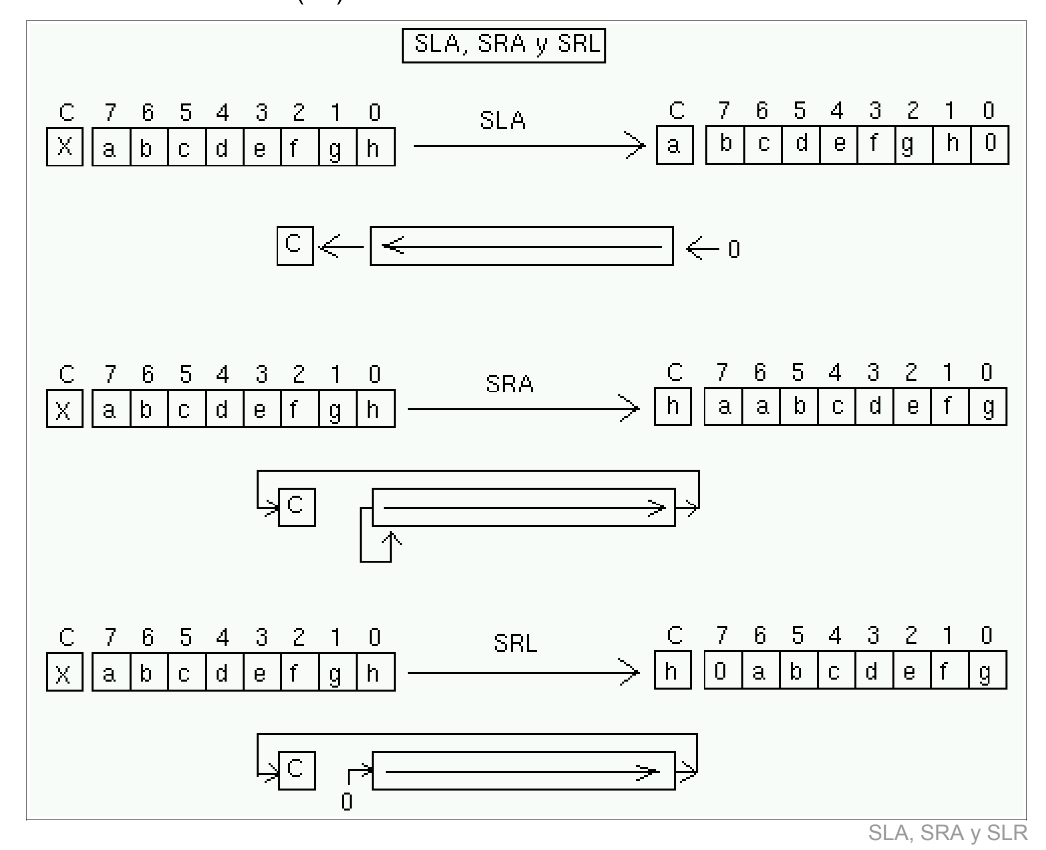
De este modo, acabamos de descubrir una manera muy sencilla y efectiva (y rápida, muy rápida para el microprocesador) de efectuar multiplicaciones y divisiones por 2, 4, 8, 16, 32, 64 y 128.

Existe una pequeña variante de SRA llamada SRL que realiza la misma acción que SRA pero que, a diferencia de esta, lo que hace es introducir un cero a la izquierda (en lugar de copiar el bit de signo). La diferencia es que SRA es un desplazamiento aritmético (tiene en cuenta el signo) y SRL es un desplazamiento lógico (simplemente desplaza los bits):

```
Bit 7 6 5 4 3 2 1 0                 C    7 6 5 4 3 2 1 0
   ----------------- -> SRL ->  ------------------------
     a b c d e f g h                h    0 a b c d e f g
```

Literalmente:

- Rotar los bits a la derecha (>>).
- El bit "h" (bit 0) se copia al Carry Flag.
- Por la izquierda entra un cero.



SLA, SRA y SLR

Nuestra ya conocida tabla:

```
                       Flags
 Instrucción         |S Z H P N C|        Significado
-----------------------------------------------------------------
 SLA s               |* * 0 P 0 *|        Shift Left Arithmetic (s=s*2)
 SRA s               |* * 0 P 0 *|        Shift Right Arithmetic (s=s/2)
 SRL s               |* * 0 P 0 *|        Shift Right Logical (s=s>>1)
```

Cabe destacar que gracias al Carry flag podremos realizar operaciones que desborden los 8 bits de que dispone un registro. Por ejemplo, supongamos que queremos realizar una multiplicación por 152 por 2. El resultado del desplazamiento sería:

```
152 = 10011000
          << 1 (*2)
    ------------
      00110000 = 48
```

¿Por qué nuestro registro acaba con un valor 48? Porque el resultado es mayor que 255, el valor máximo que podemos representar con 8 bits. Para representar el resultado (304), necesitaríamos un bit extra (9 bits) que nos daría acceso a representar números en el rango de 0 a 511. Ese bit extra es el carry flag, ya que en realidad:

```
152 =     10011000
              << 1 (*2)
    ---------------------
     1    00110000 = 304
    (C)
```

Además, gracias a la combinación de instrucciones de rotación y desplazamiento podemos realizar operaciones con registros de 16 bits. Por ejemplo, supongamos que queremos multiplicar por 2 el valor positivo que tenemos en el registro DE:

```
 SLA E
 RL  D
```

Lo que hacemos con "SLA E" es desplazar el byte más bajo del registros 16 bits DE hacia la izquierda, dejando el bit 7 de "E" en el Carry Flag, y después realizar una rotación de "D" hacia la izquierda introduciendo el carry flag de la operación anterior en el bit 0 de "D".

Registro DE original:

```
              D                        E
DE:   ---------------------   ---------------------
Bit  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00       Carry
    --------------------------------------------------     -----
      a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p         ?
```

Primero con SLA E rotamos la parte baja, metiendo el bit "i" en el Carry Flag:

```
SLA E:
              D                        E
DE:   ---------------------   ---------------------
Bit  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00       Carry
    --------------------------------------------------     -----
      a  b  c  d  e  f  g  h  j  k  l  m  n  o  p  0         i
```

Ahora con RL D rotamos D introduciendo el bit "i" en su bit 0:

RL D:

```
              D                        E
DE:   ---------------------   ---------------------
Bit  15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00       Carry
    --------------------------------------------------     -----
      b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  0            a
```

Podemos repetir la operación para multiplicar por 4, 8, 16, etc. dicho par de registros.

De igual forma, podemos realizar rotaciones de 16 bits a la derecha, haciendo el proceso inverso y comenzando primero con el byte alto:

```
SRA D
RR E
```

Las operaciones de este tipo sobre registros de 16 bits son muy importantes para realizar otro tipo de operaciones de más amplitud como multiplicaciones y divisiones.

Y, para finalizar, veamos cómo el operando destino de 16 bits puede ser un par de bytes de memoria, como en el siguiente código de ejemplo:

```
LD IX, 16384
SLA (IX)
RL (IX+01H)
```

Recordad para este ejemplo que en memoria se almacena primero el byte menos significativo de la palabra de 16 bits, y en la siguiente posición de memoria el más significativo.

OPERACIONES LOGICAS: AND, OR Y XOR

Para acabar con el artículo de hoy vamos a ver 3 operaciones a nivel de bits: AND, OR y XOR. Estas 3 operaciones lógicas se realizan entre 2 bits, dando un tercer bit como resultado:

```
  Bit  Bit   Resultado
   1    2       AND
----------------------
   1    1       1
   1    0       0
   0    1       0
   0    0       0


  Bit  Bit   Resultado
   1    2       OR
----------------------
   1    1       1
   1    0       1
   0    1       1
   0    0       0


  Bit  Bit   Resultado
   1    2       XOR
----------------------
   1    1       0
   1    0       1
   0    1       1
   0    0       0
```

Podría decirse que:

- AND es la multiplicación lógica: si cualquiera de los 2 bits es cero, el resultado es cero (0*0=0, 0*1=0, 1*0=0); dicho resultado sólo será uno cuando ambos bits sean 1 (1*1=1).
- OR es la suma lógica: si alguno de los bits es uno, el resultado es uno (1+1=1, 0+1=1, 1+0=1). Sólo obtendremos un 0 al hacer un OR entre 2 bits cuando ambos son cero.
- XOR es una operación de "O EXCLUSIVO" (exclusive OR) donde el resultado es cero cuando los 2 bits operandos son iguales, y uno cuando los 2 bits operandos son diferentes.

Ejemplos:

```
10010101 AND 0000111  = 00000101

00000101 OR  1100000  = 11000101

11000011 XOR 10011001 = 01011010
```

A la hora de realizar estas operaciones lógicas en

nuestro Z80 disponemos de 3 instrucciones cuyos nombres son, como podéis imaginar, AND, OR y XOR. Las tres tienen el mismo formato:

```
AND ORIGEN
OR  ORIGEN
XOR ORIGEN
```

Donde ORIGEN puede ser cualquier registro de 8 bits, valor inmediato de 8 bits, contenido de la memoria apuntada por [HL], o contenido de la memoria apuntada por un registro índice más un desplazamiento. El formato de la instrucción no requiere 2 operandos, ya que el registro destino sólo puede ser A.

La operación CPL, que vimos al principio de esta entrega, también se considera una operación lógica, equivalente a NOT (0->1 y 1->0).

Pero continuemos con AND, OR y XOR. Veamos algunos ejemplos de instrucciones válidas:

```
AND B
OR C
OR [HL]
XOR [IX+10]
AND 45
```

La operación realizada por estas instrucciones sería:

```
AND ORIGEN -> A = A & ORIGEN
OR  ORIGEN -> A = A | ORIGEN
XOR ORIGEN -> A = A ^ ORIGEN

( Donde & = AND, | = OR y ^ = XOR )
```

Recordemos que AND, OR y XOR son operaciones de un sólo bit, de modo que al trabajar con registros (o memoria, o valores inmediatos), en realidad estamos realizando 8 operaciones AND, OR o XOR. Por ejemplo, al hacer un AND entre los registros A y B con "AND B" (A=A&B), realizamos las siguientes operaciones:

```
Registro A:   Bit  7  6  5  4  3  2  1  0
              ----------------------------
                  A7 A6 A5 A4 A3 A2 A1 A0

Registro B:   Bit  7  6  5  4  3  2  1  0
              ----------------------------
                  B7 B6 B5 B4 B3 B2 B1 B0
```

Resultado:

```
A7 = A7 AND B7
A6 = A6 AND B6
A5 = A5 AND B5
A4 = A4 AND B4
A3 = A3 AND B3
A2 = A2 AND B2
A1 = A1 AND B1
A0 = A0 AND B0
```

Es decir, se hace una operación AND entre el bit 7 de A y el bit 7 de B, y se almacena el resultado en el bit 7 de A, y lo mismo para los bits restantes.

¿Para qué pueden servirnos estas 3 operaciones lógicas? Os aseguro que a lo largo de vuestros programas tendréis que usarlas, y mucho,

porque son operaciones muy importantes a la hora de manipular registros. Por ejemplo, supongamos que queremos eliminar los 4 bits más altos de un registro, dejándolos a cero, y dejar sin alterar el estado de los 4 bits menos significativos.

Podríamos hacer:

```
RES 7, A
RES 6, A
RES 5, A
RES 4, A
```

Pero sería mucho más sencillo:

```
AND %00001111
```

O sea, realizar la operación:

```
A = A AND 00001111b
```

Veamos un ejemplo del porqué:

```
Sea A = 10101011
valor = 00001111
      ------------ <-- Operación AND
        00001011
```

Como AND es la operación lógica de la multiplicación, al hacer un AND de A con 00001111, todos aquellos bits que son cero en 00001111 quedarán a cero en el resultado, y todos aquellos bits que son uno en 00001111 no modificarán el estado de los bits de A.

De la misma forma, por ejemplo, OR nos permite fusionar 2 cuartetos de bits:

```
Sea A = 10100000
Sea B = 00001111
      ------------ <-- Operación OR
        10101111
```

La afectación de flags de las 3 instrucciones es idéntica:

```
                           Flags
 Instrucción        |S Z H P N C|
----------------------------------
 AND s              |* * * P 0 0|
 OR s               |* * * P 0 0|
 XOR s              |* * * P 0 0|
```

Una curiosidad: XOR A es equivalente a "LD A, 0". Dejamos como ejercicio al lector comprobar por qué mediante algún ejemplo práctico.

EN LA PROXIMA ENTREGA

Veremos conceptos e instrucciones muy importantes en cualquier programa en ensamblador: qué es y cómo se usa la PILA (stack) del Spectrum, y cómo realizar cambios en el flujo del programa con sentencias condicionales equivalentes al IF/THEN/ELSE.

FICHEROS

Ejemplos de suma y resta en BCD:
http://magazinezx.speccy.org/revistas/15/src/sumaresta.txt

Ejemplo de programa en BASIC que copia la ROM en la VRAM:
http://magazinezx.speccy.org/revistas/15/src/copiarom.bas

Fichero tap del ejemplo copiarom.bas:
http://magazinezx.speccy.org/revistas/15/src/compiarom.tap

Rutina de volcado de pantalla en ASM:
http://magazinezx.speccy.org/revistas/15/src/carga.asm

Rutina de volcado de pantalla en ASM (binario):
http://magazinezx.speccy.org/revistas/15/src/carga.bin

Cargador BASIC de la rutina anterior:
http://magazinezx.speccy.org/revistas/15/src/carga.bas

Ejemplo ejecutable de carga de pantalla con volcado:
http://magazinezx.speccy.org/revistas/15/src/ejemplo.asm

Ejemplo de programa en ASM:
http://magazinezx.speccy.org/revistas/15/src/ejemplo.asm

Fichero tap del ejemplo anterior:
http://magazinezx.speccy.org/revistas/15/src/ejemplo.tap



Web del Z80:
http://www.z80.info/

Z80 Reference de TI86:
http://ti86.acz.org/z80_ref.htm

Z80 Reference de WOS:
http://www.worldofspectrum.org/faq/reference/z80reference.htm

Juego de caracteres:
http://www.worldofspectrum.org/ZXSpectrum128+3Manual/chapter8pt28.html

FAQ de Icarus Productions:
http://icarus.ticalc.org/articles/z80_faq.html

Microfichas de CM de MicroHobby:
http://www.speccy.org/trastero/cosas/Fichas/fichas.htm

Tablas de ensamblado y t-estados (pulsar en z80.txt, z80_reference.txt, z80time.txt):
http://www.ticalc.org/pub/text/z80/

Z80 Reference de WOS:
http://www.worldofspectrum.org/faq/reference/Z80reference.htm

Curso de ensamblador de z80.info:
http://www.z80.info/lesson1.htm

Pasmo:
http://www.arrakis.es/~ninsesabe/pasmo/

SROMERO


ZX
magazine